Configuration Manual

MSc Research Project
Data Analytics

Balaji Pari
Student ID: 21217394

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Balaji Pari
Student ID:	21217394
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Abubakr Siddig
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Balaji Pari
21217394

1 Introduction

This document contains the step by step process implemented in this project along with the code snippets. It explains every experiments that is conducted in this project to achieve the final accuracy.

2 Environment Setup

Table 1 shows the environment setup

Environment	Jupyter Notebook
Coding Language	Python

Table 1: Tools Used for this Project

3 Data Source

Data set for this project is taken from kaggle an open source data repository. Data is about review from readers about books. There are two data sets for this project, one data set with details of the book and other data set with rating and reviews for each book by the readers.

4 Implementation

Figure 1 shows the necessary libraries imported for this project and Figure 2 shows the code for data preparation that involves reading the two data sets, Performing basic data cleaning task like duplicate removal and null values removal. Data is then filtered from the book details data set to keep only reviews after 2017. Extracting opinions is based on the recent reviews from the readers so last six years data is considered for this project. Then the two data sets is assigned to variables for merging. After that the two data sets is merged together to form a single data frame with required columns.

Figure 3 shows the sampling of the records so that the data will not be biased towards a rating and only the required columns are taken and the columns are renamed in the final dataframe

```
#Importing the required modules
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re
import nltk
import string
import contractions
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from gensim import corpora, models
from wordcloud import WordCloud
from nltk.tokenize import word_tokenize, sent_tokenize
from bs4 import BeautifulSoup
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
import warnings
warnings.filterwarnings("ignore")
from textblob import TextBlob
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.preprocessing import LabelBinarizer
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
import gensim
from gensim import corpora
import pyLDAvis.gensim_models as gensimvis
```

Figure 1: Importing Necessary Libraries

```
#Reading the csv file
df = pd.read_csv('Books_rating.csv')
```

```
df_1 = pd.read_csv('books_data.csv',low_memory=False)
```

```
#Droping the Null Values
df_1 = df_1.dropna(how='all')
df_1 = df_1.dropna(subset=['publishedDate'])
df = df.dropna(subset=['User_id'])
df = df.dropna(subset=['review/summary'])
df = df.dropna(subset=['review/text'])
```

```
#Droping Duplicates
df.drop_duplicates(subset=['review/summary','review/text'], inplace=True)
```

```
#Filtering the data to get records from 2017
df_1['publishedDate'] = pd.to_datetime(df_1['publishedDate'],format='%d-%m-%Y', dayfirst=True)
start_date = pd.to_datetime('01-01-2017')
end_date = pd.to_datetime('01-12-2023')
df_1 = df_1[(df_1['publishedDate'] >= start_date) & (df_1['publishedDate'] <= end_date)]
```

```
#Assigning dataframe to a new variable
Reviews = df
Books = df_1
```

```
#joining the two data frames
Book_Reviews = pd.merge(Books,Reviews, on = 'Title')
```

Figure 2: Data Preparation - 1

```python
#fetching equal number of records for each rating to avoid bias
df_five = df_five[:8500]
df_four = df_four[:7500]
df_three = df_three[:9954]
df_two = df_two[:5581]
df_one = df_one[:7669]
```

```python
#merging all the dataframes into single dataframne
Book_Reviews = pd.concat([df_one, df_two, df_three, df_four, df_five], axis=0)
```

```python
#Reset index values
Book_Reviews.reset_index(drop=True, inplace=True)
```

```python
#removing extra spaces in column names
Book_Reviews.columns = Book_Reviews.columns.str.strip()
```

```python
#Keeping only the required columns
Book_Reviews=Book_Reviews[['Title','description','User_id','review/score','review/summary',\
                           'review/text','authors','categories','publisher']]
```

```python
#Renaming column names
Book_Reviews.rename(columns={'review/score':'Rating','review/text':'Review','categories':'Genre','authors':'Author'\
                             ,'description':'Description','User_id':'User','review/summary':'Review Summary'},inplace=True)
```

Figure 3: Data Preparation - 2

Figure 4 shows the code for text cleaning process which involves conversion upper to lower case, removal of special characters, removal of extra spaces, removal of punctuation's etc.

```python
#funcation for normalizaing the text
def normalize_text(text):
    text = text.lower()
    text = contractions.fix(text)
    return text
```

```python
#Applying the normalize text function for reviews and review summary
Book_Reviews['Clean_Review'] = Book_Reviews['Review'].apply(normalize_text)
Book_Reviews['Clean Review Summary'] = Book_Reviews['Review Summary'].apply(normalize_text)
```

```python
#Text cleaning function
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation
    and remove words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

```python
#Applying the text cleaning function for reviews and review summary
Book_Reviews['Clean_Review'] = Book_Reviews['Clean_Review'].apply(lambda x:clean_text(x))
Book_Reviews['Clean Review Summary'] = Book_Reviews['Clean Review Summary'].apply(lambda x:clean_text(x))
```

Figure 4: Text Cleaning

Once text is cleaned stop words are removed from the review text and Figure 5 shows the code for stop word removal using the English language model. After stop words removal, lemmetization is performed to retain the base form of a word. Figure 6 shows the code for lemmetization.

After the lemmetization process, a new column is created called sentiment. It is created based on rating column and to label them as negative, neutral and positive. After label them with the sentiments, label encoding is performed to convert them into numerical values as 0,1 and 2. Figure 7 shows the code for label encoding process. Figure 8 shows the sample data set after all the preprocessing steps.

```python
#Function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    return text
```

```python
#Applying the special characters function for reviews, review summary and genre
Book_Reviews['Clean_Review']=Book_Reviews['Clean_Review'].apply(remove_special_characters)
Book_Reviews['Genre']=Book_Reviews['Genre'].apply(remove_special_characters)
Book_Reviews['Clean Review Summary']=Book_Reviews['Clean Review Summary'].apply(remove_special_characters)
```

```python
#Asigning the stop word english corpus to a variable
#stop=set(stopwords.words('english'))
stopword_list=nltk.corpus.stopwords.words('english')
```

```python
#Function for removing stop words
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
```

```python
#Defining the tokenizaer
tokenizer=ToktokTokenizer()
```

```python
#Applying the stop word function for reviews and review summary
Book_Reviews['Clean_Review']=Book_Reviews['Clean_Review'].apply(remove_stopwords)
Book_Reviews['Clean Review Summary']=Book_Reviews['Clean Review Summary'].apply(remove_stopwords)
```

Figure 5: Stop Words Removal

```python
# Loading the English language model
nlp = spacy.load('en_core_web_sm')
```

```python
#Defining for lemmetization
def get_lemmas(text):
    lemmas = []
    doc = nlp(text)
    for token in doc:
        if ((token.is_stop == False) and (token.is_punct == False)) and (token.pos_ != 'PRON'):
            lemmas.append(token.lemma_)
    return lemmas
```

```python
#Applying the lemmetization function for reviews and review summary
Book_Reviews['Clean_Review']=Book_Reviews['Clean_Review'].apply(get_lemmas)
Book_Reviews['Clean Review Summary']=Book_Reviews['Clean Review Summary'].apply(get_lemmas)
```

```python
#joining the tokenized words to form a sentence again
Book_Reviews['Clean_Review']= [' '.join(map(str, l)) for l in Book_Reviews['Clean_Review']]
Book_Reviews['Clean Review Summary']= [' '.join(map(str, l)) for l in Book_Reviews['Clean Review Summary']]
```

Figure 6: Lemmetization

```python
def map_to_sentiment(rating):
    if rating >= 4:
        return 'Positive'
    elif rating == 3:
        return 'Neutral'
    else:
        return 'Negative'
```

```python
# Apply the function to create a new column with sentiment labels
Book_Reviews['Sentiment']= Book_Reviews['Rating'].apply(map_to_sentiment)
```

```python
label_encoder = LabelEncoder()
Book_Reviews['Sentiment'] = label_encoder.fit_transform(Book_Reviews['Sentiment'])
```

Figure 7: Label Encoding

Title	Description	User	Rating	Review Summary	Review	Author	Genre	publisher	Clean_Review	Clean Review Summary	Sentiment
Cruel and Unusual (G K Hall Large Print Book S...	Wanneer er in dit achtste deel in de Kay Scarp...	A2Y34QTG3XDBIA	1	2/3 of a great book, then... blah	I listened to the audio version of "Cruel...	['Patricia Cornwell']	Fiction	Luitingh Sijthoff	listen audio version quotcruel amp unusualquot...	[book, blah]	0
Cruel and Unusual (G K Hall Large Print Book S...	Wanneer er in dit achtste deel in de Kay Scarp...	A2W3TP5CDRH29W	1	I guess it was a mystery	In a word, Lacking. Her style of writing left ...	['Patricia Cornwell']	Fiction	Luitingh Sijthoff	word lacking style writing leave unable apprec...	[guess, mystery]	0
Cruel and Unusual (G K Hall Large Print Book S...	Wanneer er in dit achtste deel in de Kay Scarp...	A3HEV7S1SS2HID	1	Inproper Advertising	I needed to replace a lost CD set of the libra...	['Patricia Cornwell']	Fiction	Luitingh Sijthoff	need replace lose cd set library cd book cruel...	[inproper, advertising]	0
Eco-Terrorism & Eco-Extremism Against Agriculture	This book scrutinizes the growth of the 'eco-t...	A2XMKGUFQSHXHH	1	A Self-Published Term Paper, Nothing More	I gave 1 star only because Amazon doesn't let ...	['Gerry Nagtzaam']	Ecoterrorism	Edward Elgar Publishing	give star amazon let zero case negative starss...	[selfpublishe, term, paper]	0
Island	Anne Cholawo was a typical 80s career girl wor...	A1BZRECABLFF8G	1	Self-Indulgent Drivel	Prior to reading this I thought that maybe as	['Anne Cholawo']	Biography Autobiography	Birlinn Publishers	prior reading think maybe huxley get old impro...	[selfindulgent, drivel]	0

Figure 8: Preprocessed Sample Dataset

4.1 Experiment 1 - Count Vectorization and ratings as target column

Once the preprocessing is done, X and Y variables are declared with independent and dependent variable respectively where X is the cleaned review text column and Y is the rating column. Feature extraction performed using count vectorization technique and base models are implemented using Naive Bayes, Decision Tree and Random Forest. Figure 9 shows the code for count vectorization on cleaned review text column and Figure 10 shows the example for implementation of base model using Naive Bayes algorithm.

```
x = Book_Reviews['Clean_Review']
y= Book_Reviews['Rating']
```

```
#Defining function for text processing while convert them into vectors
def text_process(text):
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
```

Count vectorization on review column and running the model with rating as target variable

```
# Converting the word into vectors
vocab = CountVectorizer(analyzer=text_process).fit(x)
r0 = x[0]
vocab0 = vocab.transform([r0])
```

```
x = vocab.transform(x)
#Shape of the matrix:
print("Shape of the sparse matrix: ", x.shape)
#Non-zero occurences:
print("Non-Zero occurences: ",x.nnz)
# DENSITY OF THE MATRIX
density = (x.nnz/(x.shape[0]*x.shape[1]))*100
print("Density of the matrix = ",density)
```

```
Shape of the sparse matrix:  (22855, 109072)
Non-Zero occurences:  1459712
Density of the matrix =  0.0585556171759336265
```

```
# splitting the dataset into test and train
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=101)
```

Figure 9: Count Vectorization and Rating as Target

Figure 10 shows the implementation code for decision tree model and Table 2 shows the accuracy for all the three base models using count vectorization as feature extraction technique and rating as target column

```python
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
preddt = dt.predict(x_test)
print("Confusion Matrix for Decision Tree:")
print(confusion_matrix(y_test,preddt))
print("Score:",round(accuracy_score(y_test,preddt)*100,2))
print("Classification Report:",classification_report(y_test,preddt))
```

Figure 10: Example code for Decision Tree model

Model	Accuracy
Naive Bayes	49.32
Decision Tree	33.91
Random Forest	45.87

Table 2: Experiment 1- Accuracy

4.2 Experiment 2 - Count Vectorization and Sentiments as target column

In experiment 2, count vectorization is implemented as feature extraction technique and models are implemented with sentiment class as target column. Figure 11 shows the code implementation for experiment 2 and Table 3 shows the accuracy of all the three models and it can be seen that Naive Bayes algorithm performs the best.

Count vectorization on review column sentiment class as target column and running the models

```python
x_1 = Book_Reviews['Clean_Review']
y_1= Book_Reviews['Sentiment']
```

```python
# Converting the word into vectors
vocab_1 = CountVectorizer(analyzer=text_process).fit(x_1)
r0_1 = x_1[0]
vocab0_1 = vocab_1.transform([r0_1])
```

```python
x_1 = vocab_1.transform(x_1)
#Shape of the matrix:
print("Shape of the sparse matrix: ", x_1.shape)
#Non-zero occurences:
print("Non-Zero occurences: ",x_1.nnz)
# DENSITY OF THE MATRIX
density = (x_1.nnz/(x_1.shape[0]*x_1.shape[1]))*100
print("Density of the matrix = ",density)
```

```
Shape of the sparse matrix:  (39204, 140791)
Non-Zero occurences:  2217989
Density of the matrix =  0.040184087777307295
```

```python
# splitting the dataset into test and train
x_train,x_test,y_train,y_test = train_test_split(x_1,y_1,test_size=0.2,random_state=101)
```

Figure 11: Count Vectorization and Sentiment as Target

Model	Accuracy
Naive Bayes	66.22
Decision Tree	48.46
Random Forest	62.57

Table 3: Experiment 2 - Accuracy

4.3 Experiment 3 - TF-IDF Vectorization and Sentiments as target column

In experiment 3, TF-IDF is used as feature extraction technique and sentiment column is used as target column. Models are implemented to predict the sentiment class. Figure 12 shows the implementation code for TF-IDF vectorization and Naive Bayes algorithm. Table 4 shows the accuracy for all the three models and it looks like naive bayes performs the best. Compaer to the previous experiment there is no big change in the accuracy of the models.

TF_IDF Vectorization on review column and running the model with sentiment class as target variable

```
tfidf_vectorizer = TfidfVectorizer()

x_1 = Book_Reviews['Clean_Review']
y_1= Book_Reviews['Sentiment']

x_1 = tfidf_vectorizer.fit_transform(x_1)

# splitting the dataset into test and train
x_train,x_test,y_train,y_test = train_test_split(x_1,y_1,test_size=0.2,random_state=101)

# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
mnb.fit(x_train,y_train)
predmnb = mnb.predict(x_test)
print("Confusion Matrix for Multinomial Naive Bayes:")
print(confusion_matrix(y_test,predmnb))
print("Score:",round(accuracy_score(y_test,predmnb)*100,2))
print("Classification Report:",classification_report(y_test,predmnb))
```

Figure 12: TF-IDF Vectorization and Sentiment as Target

Model	Accuracy
Naive Bayes	66.22
Decision Tree	48.46
Random Forest	62.57

Table 4: Experiment 3 - Accuracy

4.4 Experiment 4 - TF-IDF Vectorization and Sentiments as target column and performed Over Sampling using SMOTE

In this experiment, same like previous TF-IDF vectorization is implemented and sentiment column set as target. Along with that over sampling using SMOTE is implemented to balance the sentiments as there is slight imbalance when label encoding is performed and to overcome that oversampling is performed to match the sentiment classes. Figure

13 shows the implementation code for performing over sampling using SMOTE and Figure 14 shows the hyperparameter tuning for Naive Bayes algorithm and it can be seen that for alpha value 0.1 it has maximum mean test score.

```
from imblearn.over_sampling import SMOTE
import pandas as pd

x = Book_Reviews['Clean_Review']
y = Book_Reviews['Sentiment']

# Initialize and fit the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=50000)
X_text_vectorized = tfidf_vectorizer.fit_transform(x)

# Apply SMOTE to oversample the minority classes
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_text_vectorized, y)

# Split the resampled data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB(alpha=0.5)
mnb.fit(x_train,y_train)
predmnb = mnb.predict(x_test)
print("Confusion Matrix for Multinomial Naive Bayes:")
print(confusion_matrix(y_test,predmnb))
print("Score:",round(accuracy_score(y_test,predmnb)*100,2))
print("Classification Report:",classification_report(y_test,predmnb))
```

Figure 13: Over Sampling

```
#Hyperparameter Tuning
alphas = [0.1, 0.5, 1.0, 1.5, 2.0]
param_grid = {'alpha': alphas}
# Create the Grid Search object
grid_search = GridSearchCV(mnb, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)
results_df = pd.DataFrame(grid_search.cv_results_)
# Extract only the hyperparameters and their corresponding mean test score
selected_columns = ['params','mean_test_score']
results_selected = results_df[selected_columns]
# Print the Grid Search results in a DataFrame
print("Grid Search Results:")
print(results_selected)
# Get the best hyperparameters found during the search
best_alpha = grid_search.best_params_['alpha']
print("Best alpha:", best_alpha)
```

```
Grid Search Results:
             params  mean_test_score
0  {'alpha': 0.1}         0.691172
1  {'alpha': 0.5}         0.691146
2  {'alpha': 1.0}         0.685156
3  {'alpha': 1.5}         0.678906
4  {'alpha': 2.0}         0.672474
Best alpha: 0.1
```

Figure 14: Hyperparameter Tuning for Naive Bayes

Table 5 shows the accuracy for the Naive Bayes model with the hyperparameter tuning and over sampling of data. From this it can be seen that experiment 4 have the maximum accuracy of 68.85 percentage compared to other experiments. Finally Multinomial Model with alpha value of 0.1 is considered as the final model for the proposed methodology. Figure 15 shows the confusion matrix for the Naive Bayes model and Figure 16 shows the Classification report of Naive Bayes.

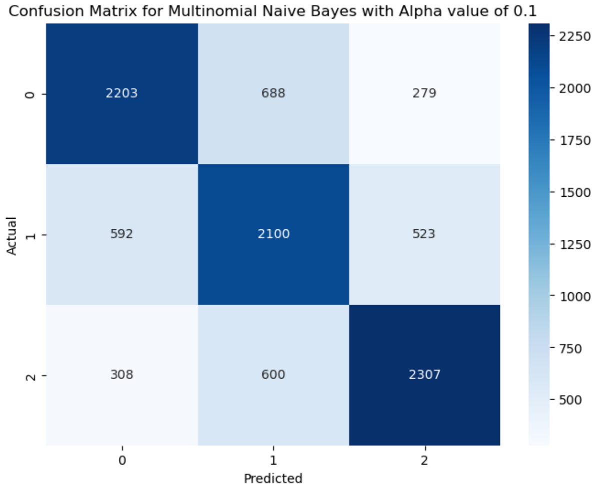

Figure 15: Confusion Matrix of Naive Bayes

```
Confusion Matrix for Multinomial Naive Bayes:
[[2203  688  279]
 [ 592 2100  523]
 [ 308  600 2307]]
Score: 68.85
Classification Report:               precision    recall  f1-score   support

           0       0.71      0.69      0.70      3170
           1       0.62      0.65      0.64      3215
           2       0.74      0.72      0.73      3215

    accuracy                           0.69      9600
   macro avg       0.69      0.69      0.69      9600
weighted avg       0.69      0.69      0.69      9600
```

Figure 16: Classification report of Naive Bayes

4.5 Implementation of topic modelling using LDA(Latent Dirichlet Allocation)

Topic modelling is implemented on negative and positive reviews to understand about the opinions of the readers on different genres. Figure 17 shows the code implementation for LDA and Figure 18 shows the implementation code for calculation of cosine similarity in order to assign the extracted topics to each review and it is done by a look up table and In Figure 18 it can be seen that lookup table is created for the extracted topics and topic name is given to the each topic based on the key words. Figure 19 shows the frequency of the words from each topics.

```
#Creating the dictionary
dictionary = corpora.Dictionary(Fiction['Clean Review Summary'])
```

```
# Each tokenized words has been assigned index value and thier count in corpus
doc_term_matrix = Fiction['Clean Review Summary'].apply(lambda x: dictionary.doc2bow(x))
```

```
# corpus requires document term matrix
# num_topics is used to define number of topics to create from corpus
# id2word requires mapping of words
# passes is used to define number of iterations
Lda = gensim.models.ldamodel.LdaModel
ldamodel_Fiction = Lda(corpus=doc_term_matrix, num_topics=8, id2word=dictionary, passes=10,random_state=45)
clear_output()
```

Figure 17: Implementation of LDA

	Topic_Number	Top_Keywords	Topic_Name
0	0	[love, version, story, fiction, premise, movie...	Rework and disconnection
1	1	[boring, edition, review, bore, dull, plot, fa...	Boring Story and plot
2	2	[series, reader, rise, fun, sun, quality, tell...	Writing and Finish
3	3	[way, lot, cliche, execution, repeat, edition,...	Long and Shallow
4	4	[write, end, disappoint, author, buick, drivel...	dull and disappointment
5	5	[story, finish, work, miss, try, lack, awful, ...	lame and mislead of characters
6	6	[want, ok, okay, format, return, think, error,...	Rework on finishing
7	7	[disappointment, lame, light, pointless, lead,...	grammar and printing
8	8	[novel, time, waste, writer, plot, romance, co...	Plot and content
9	9	[character, star, worth, money, effort, waste,...	ideas and narration

```
# Step 1: Convert 'Top_Keywords' into a list of strings
topic_lookup_data['Top_Keywords'] = topic_lookup_data['Top_Keywords'].apply(lambda x: ' '.join(x))
```

```
Fiction['Clean Review Summary'] = Fiction['Clean Review Summary'].apply(lambda x: ' '.join(x))
```

```
vectorizer = CountVectorizer()
reviews_vectorized  = vectorizer.fit_transform(Fiction['Clean Review Summary'])
```

```
topic_assignments = []
for review_vector in reviews_vectorized:
    similarities = cosine_similarity(review_vector, vectorizer.transform(topic_lookup_data['Top_Keywords']))[0]
    assigned_topic = topic_lookup_data['Topic_Name'][similarities.argmax()]
    topic_assignments.append(assigned_topic)
```

Figure 18: Cosine Similarity

4.6 Result

Figure 20 and 21 shows the frequency of each topic in the negative and positive reviews and it can be seen that the negative and positive opinions are extracted from the reviews about fiction genre.

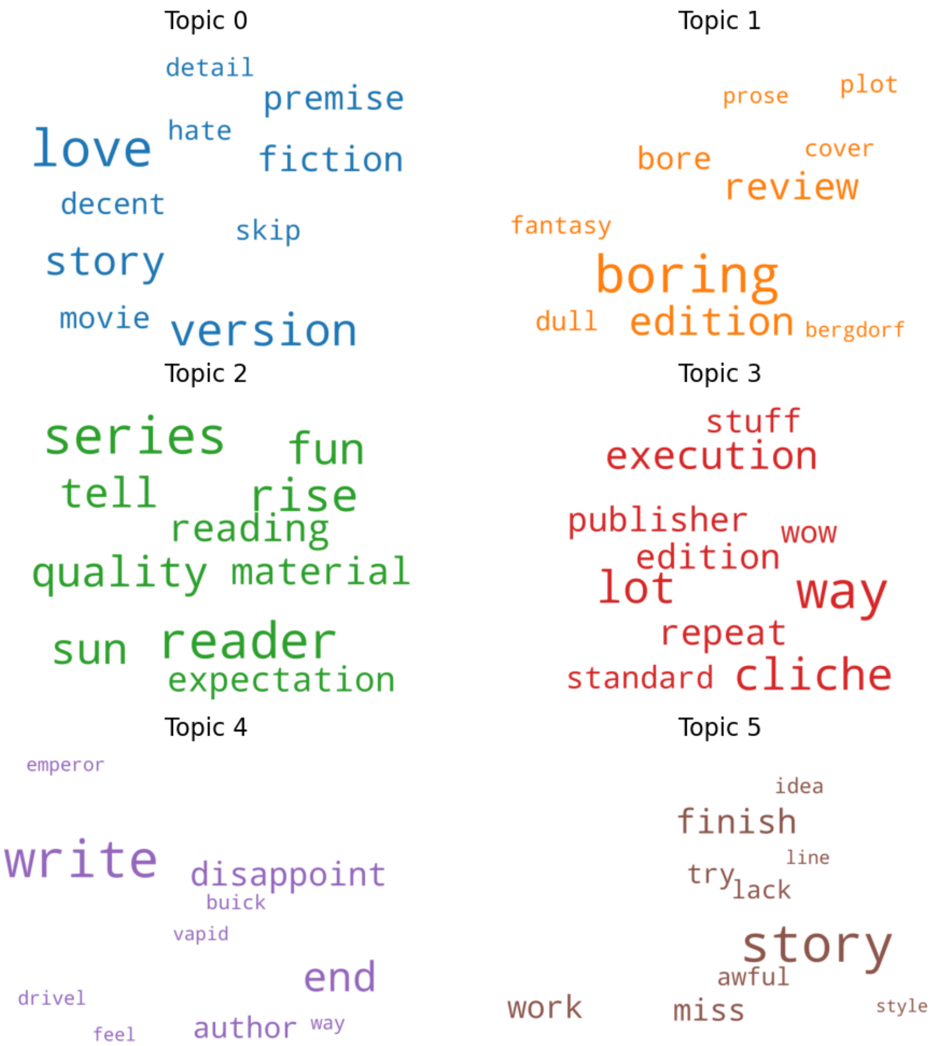

Figure 19: Topic and its Key words

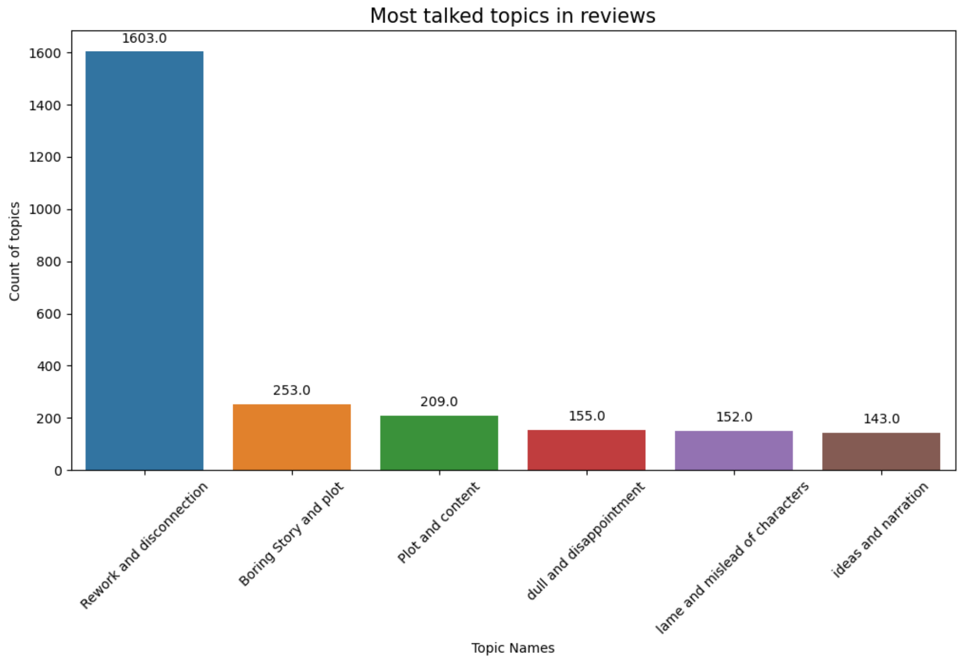

Figure 20: Negative Opinion - Fiction Genre

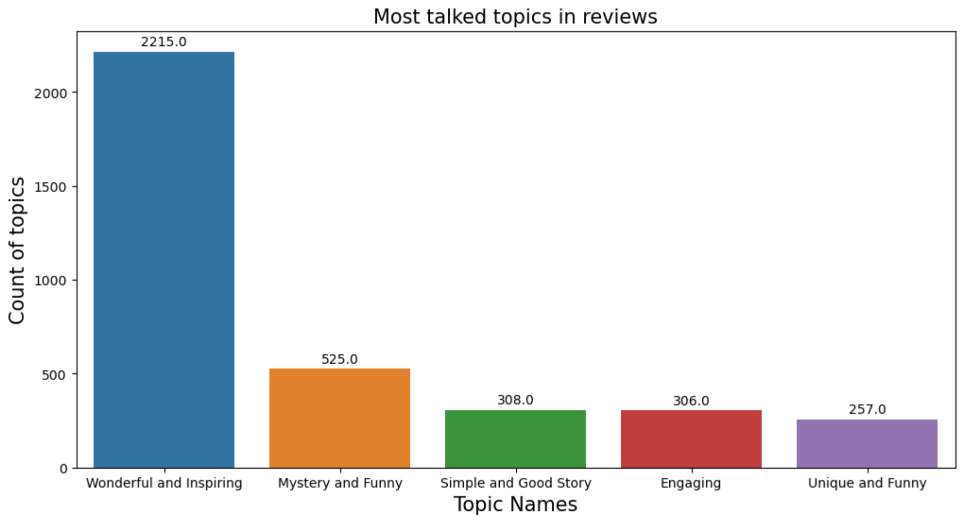

Figure 21: Positive Opinions - Fiction Genre