Hausarbeit Maschinelles Lernen MADS23oB Heidi Gehring, Lea Kleemann

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende akademische Abschlussarbeit selbstständig und ohne fremde Hilfe angefertigt habe.
Alle Textstellen, die ich wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Quellen übernommen habe, wurden von mir als solche gekennzeichnet.

Heidi Gehring

Lea Kleemann

Problem Statement

Dataset for the analysis

In this assignment the following Dataset is used. It was initially suggested to be used to create a recommender system for books.

The data set contains information about books, their ratings, and users who rated the books and is available on Kaggle. The information contains characteristics such as book-id, title, author, ISBN, genre, ratings and user-ids.

Focus of the analysis

In this assignment it will be evaluated whether the available information is suitable to predict how a given user would rate a specific book.

Link to the Repository

The Github repsitory that was used for this assignment can be found here

This notebook was executed in Python 3.8.x, because of the tensorflow dependency. Some code was originally written and executed in Python 3.12.x.

Solution approach and project outline

The objective of this assignment is to train a model that is able to predict the rating of a book by a given user analogue to the "Predicted Rating" in the file "collaborative_books_df.csv". It is not the objective to create a book recommender system.

Throughout the assignment, both classification as well as regression algomrithms and methods will be used to analyse the use case. The "Actual Ratings" are used as the target variable. As can be seen both as a finite number of categories as well as numeric values, the previous mentioned algorithms seem suitable.

This assignment is based on the initial hyptheses, that the users can be clustered into well separated groups based on what they are reading and how they are rating those books. It was the plan that those clusters shall then be used as feature input for the prediction modells for the ratings.

Additionally, some metadata about the books needs to be transformed. The focus was on the features "description" which is the blurbs of the books as well as the genres, where each book is mapped to 1 or more genres. For the blurbs, topic modeling approaches are being explored, for the genres the one-hot-encoding is being used.

Data exploration

In [1]:
```
# combining all imports for the notebook
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
import pathlib2 as pathlib
import numpy as np

from gensim.models.coherencemodel import CoherenceModel
import gensim.corpora as corpora
from gensim.parsing.preprocessing import STOPWORDS
from gensim.utils import tokenize
import re

import joblib
from optuna import Trial, create_study
from optuna.pruners import SuccessiveHalvingPruner

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.preprocessing import normalize
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures

from sklearn.decomposition import LatentDirichletAllocation
from sklearn.decomposition import NMF
from sklearn.decomposition import PCA

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV

from sklearn.impute import SimpleImputer

from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso, Ridge
from sklearn.feature_selection import RFE

from sklearn.ensemble import RandomForestRegressor

from sklearn.neighbors import KNeighborsRegressor
from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import silhouette_score
from sklearn.metrics import adjusted_rand_score

from sklearn import svm
```

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier

from sklearn.neural_network import MLPRegressor

import tensorflow as tf

randomstate=313

# treat all python warnings as lower-level "ignore" events
import warnings
warnings.filterwarnings("ignore")
```

```
c:\Users\lekle\anaconda3\envs\tf-env\lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
# define base paths
cwd=pathlib.Path.cwd() #current working directory
datadirpath=cwd.joinpath("data") # data drectory unter current working directory
rawdatapath=datadirpath.joinpath("raw")
```

Explore collaborative_books_df.csv

In [3]:
```
reviewdf=pd.read_csv(rawdatapath.joinpath("collaborative_books_df.csv"))
reviewdf=reviewdf.drop([reviewdf.columns[0]],axis=1) # removed unnamed index column
print(reviewdf.columns)
```

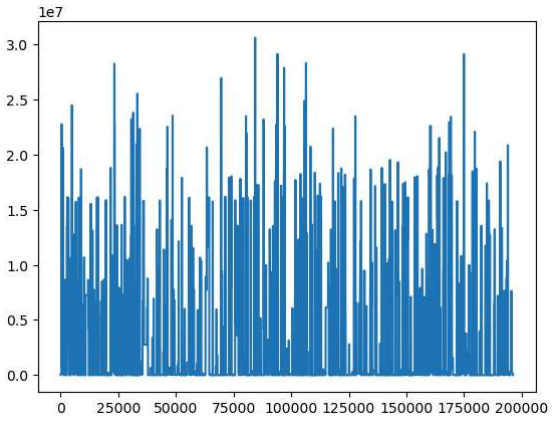
```
Index(['title', 'book_id', 'user_id_mapping', 'book_id_mapping',
       'Predicted Rating', 'Actual Rating'],
      dtype='object')
```

In [4]:
```
# print basic information
print(f"Number of observations: {len(reviewdf)}")
print(f"Number of unique book ids: {len(reviewdf['book_id'].unique())}")
print(f"Number of unique user ids: {len(reviewdf['user_id_mapping'].unique())}")
print(f"Average actual rating: {reviewdf['Actual Rating'].mean()}")
print(f"Average predicted rating: {reviewdf['Predicted Rating'].mean()}")
```

```
Number of observations: 196296
Number of unique book ids: 898
Number of unique user ids: 66909
Average actual rating: 3.922902147776827
Average predicted rating: 3.8966672779883433
```

In [5]:
```
reviewdf["book_id"].plot() # not conclusive, try grouped by id, count as aggregate metric
```

Out[5]: <Axes: >



In [6]:
```
num_books=len(reviewdf.book_id_mapping.unique())
print(f'Number of unique books: {num_books}') #898 --> that means we do not have data for all books
```

```
Number of unique books: 898
```

Explore user ids

In [7]:
```
# groupby user id
reviewdf_userids=reviewdf[["user_id_mapping","book_id"]].groupby(by="user_id_mapping").count()
reviewdf_userids=reviewdf_userids.reset_index()
reviewdf_userids=reviewdf_userids.rename(columns={"book_id":"book_count"})
```

In [8]:
```
print(f"(repeat) Number of users: {len(reviewdf_userids)}")
print(f"Max number of books revied per user: {reviewdf_userids.book_count.max()}")
print(f"Min number of books revied per user: {reviewdf_userids.book_count.min()}")
print(f"Number of users with 1 review: {len(reviewdf_userids.query('book_count == 3'))}")
print(f"... in %: {round(len(reviewdf_userids.query('book_count == 3'))/len(reviewdf_userids),2)*100}%") # 18% is suboptimal --> prediction of score for single user is probably r
print(f"Number of users with less than 6 reviews: {len(reviewdf_userids.query('book_count < 6'))}")
print(f"... in %: {round(len(reviewdf_userids.query('book_count < 6'))/len(reviewdf_userids),2)*100}%") # 90% is suboptimal
```

```
(repeat) Number of users: 66909
Max number of books revied per user: 22
Min number of books revied per user: 1
Number of users with 1 review: 12202
... in %: 18.0%
Number of users with less than 6 reviews: 59981
... in %: 90.0%
```

Explore book ids

In [9]:
```
# groupby book_id
reviewdf_bookids=reviewdf[["book_id","user_id_mapping"]].groupby(by="book_id").count()
reviewdf_bookids=reviewdf_bookids.reset_index()
reviewdf_bookids=reviewdf_bookids.rename(columns={"user_id_mapping":"user_count"})
```

In [10]:
```
print(f"(repeat) Number of books: {len(reviewdf_bookids)}")
print(f"Max number of users reviewd per book: {reviewdf_bookids.user_count.max()}")
print(f"Min number of users reviewd per book: {reviewdf_bookids.user_count.min()}")
print(f"Number of books with 1 review: {len(reviewdf_bookids.query('user_count == 3'))}")
print(f"... in %: {round(len(reviewdf_bookids.query('user_count == 3'))/len(reviewdf_bookids),2)*100}%") # 18% is suboptimal --> prediction of score for single user is probably r
```

```
print(f"Number of books with less than 6 reviews: {len(reviewdf_bookids.query('user_count < 6'))}")
print(f"... in %: {round(len(reviewdf_bookids.query('user_count < 6'))/len(reviewdf_bookids),2)*100}%") # 90% is suboptimal
```

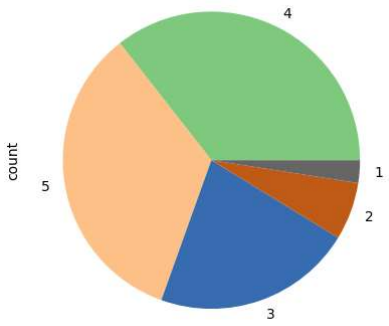
```
(repeat) Number of books: 898
Max number of users reviewd per book: 1296
Min number of users reviewd per book: 36
Number of books with 1 review: 0
... in %: 0.0%
Number of books with less than 6 reviews: 0
... in %: 0.0%
```

Explore Ratings

In [11]:
```
reviewvaluecountsdf=reviewdf["Actual Rating"].value_counts()
reviewssum=reviewvaluecountsdf.sum()
reviewvaluecountsdf["percentage"]=reviewvaluecountsdf.transform(func=lambda x:x/reviewssum*100,axis=0) #50+% at rating of more than 3

predvaluecountsdf=reviewdf["Predicted Rating"].value_counts()
predvaluecountsdf["percentage"]=predvaluecountsdf.transform(func=lambda x:x/reviewssum*100,axis=0)

bins=[1,2,3,4,5,6] #
hist, edges = np.histogram(reviewdf["Predicted Rating"],bins=bins,density=False)
predbinsvaluecountsdf=pd.DataFrame(hist,index=bins[:-1],columns=["PredictedRating_count"])
predbinsvaluecountsdf["percentage"]=predbinsvaluecountsdf.transform(func=lambda x:x/reviewssum*100,axis=0)
```

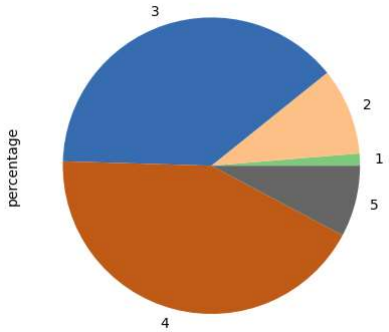
In [12]:
```
reviewvaluecountsdf["percentage"].plot(kind='pie',colormap='Accent')
```

Out[12]: <Axes: ylabel='count'>



In [13]:
```
predbinsvaluecountsdf.percentage.plot(kind='pie',colormap='Accent')
```

Out[13]: <Axes: ylabel='percentage'>



In [14]:
```
print(f"Sum of reviews: {reviewssum}")

print(f"(repeat) Average actual rating: \n {reviewdf['Actual Rating'].mean()}")
print(f"Max actual rating: {reviewdf['Actual Rating'].max()}")
print(f"Min actual rating: {reviewdf['Actual Rating'].min()}")
print(f"Overview of Rating occurence: {reviewvaluecountsdf}")
print(f"... in percentage: \n {reviewvaluecountsdf['percentage']}") # comparatively few bad reviews
print("_________________ \n")
print(f"(repeat) Average predicted rating: {reviewdf['Predicted Rating'].mean()}")
print(f"Max predicted rating: {reviewdf['Predicted Rating'].max()}")
print(f"Min predicted rating: {reviewdf['Predicted Rating'].min()}")
# print(f"Overview of Rating occurence: {predvaluecountsdf}") # continuous values
# print(f"... in percentage: \n {predvaluecountsdf["percentage"]}") # continuous values, try binning
print(f"Overview of Rating occurence: \n {predbinsvaluecountsdf['PredictedRating_count']}")
print(f"... in percentage: \n {predbinsvaluecountsdf['percentage']}")
```

```
Sum of reviews: 196296
(repeat) Average actual rating:
 3.922902147776827
Max actual rating: 5
Min actual rating: 1
Overview of Rating occurence: Actual Rating
4                                                   69813
5                                                   66663
3                                                   42680
2                                                   12303
1                                                    4837
percentage    Actual Rating
4    35.565167
5    33.960447
3 ...
Name: count, dtype: object
... in percentage:
 Actual Rating
4    35.565167
5    33.960447
3    21.742674
2     6.267575
1     2.464136
Name: count, dtype: float64
_________________

(repeat) Average predicted rating: 3.8966672779883433
Max predicted rating: 5.0
Min predicted rating: 1.0
Overview of Rating occurence:
 1     2588
2    18588
3    76042
4    83816
5    15262
Name: PredictedRating_count, dtype: int64
... in percentage:
 1     1.318417
2     9.469373
3    38.738436
4    42.698781
5     7.774993
Name: percentage, dtype: float64
```

exploring collaborative_book_metadata.csv

In [15]:
```
booksdf=pd.read_csv(rawdatapath.joinpath('collaborative_book_metadata.csv'))
booksdf=booksdf.drop([booksdf.columns[0]],axis=1) # removed unnamed index column
print(booksdf.columns)
booksgroupbyauthorsdf=booksdf[["name","book_id_mapping"]].groupby(by="name").count()
booksgroupbyauthorsdf=booksgroupbyauthorsdf.reset_index()
booksgroupbyauthorsdf=booksgroupbyauthorsdf.rename(columns={"book_id_mapping":"book_count"})
```

```
Index(['book_id', 'title', 'image_url', 'url', 'num_pages', 'ratings_count',
       'description', 'genre', 'name', 'book_id_mapping'],
      dtype='object')
```

In [16]:
```
booksgroupbyauthorsdf.sort_values(by="book_count",ascending=False).head()
```

Out[16]:

	name	book_count
7	Charlaine Harris	3
21	Fiona Staples	3
11	Christopher Paolini	3
22	George R.R. Martin	2
67	Robert Jordan	2

In [17]:
```
print(f"Number of books: {len(booksdf)}")
print("_____________________ \n")
print(f"Max number of pages: {booksdf.num_pages.max()}")
print(f"Avg number of pages: {booksdf.num_pages.mean()}")
print(f"Min number of pages: {booksdf.num_pages.min()}")
print("_____________________ \n")
print(f"Max number of ratings: {booksdf.ratings_count.max()}")
print(f"Avg number of ratings: {booksdf.ratings_count.mean()}")
print(f"Min number of ratings: {booksdf.ratings_count.min()}")
print("_____________________ \n")
print(f"Number of unique authors: {len(booksdf.name.unique())}")
print(f"Authors with most books: {booksgroupbyauthorsdf.sort_values(by='book_count',ascending=False).head()}") #max =3, not al lot of books per author, maybe not predictive enou
```

```
Number of books: 96
_____________________

Max number of pages: 4100
Avg number of pages: 388.96875
Min number of pages: 26
_____________________

Max number of ratings: 1128913
Avg number of ratings: 235801.38541666666
Min number of ratings: 50020
_____________________

Number of unique authors: 83
Authors with most books:                    name  book_count
7      Charlaine Harris           3
21        Fiona Staples           3
11  Christopher Paolini           3
22   George R.R. Martin           2
67        Robert Jordan           2
```

explore genres

In [18]:
```
# split and process genres
booksdf['genre']=booksdf.genre.apply(lambda x: x.replace("'","")) # remove '
booksdf['genre']=booksdf.genre.apply(lambda x: x.replace("-","")) # revome -
booksdf['genre']=booksdf.genre.apply(lambda x: x.replace(" ","")) # remove spaces
booksdf['genre']=booksdf.genre.apply(lambda x: x[1:-1].split(',')) # split into list
```

In [19]:
```
# get number of genres
booksdf["num_genres"]=booksdf.genre.apply(lambda x: len(x))
```

In [20]:
```
print(f"Max number of genres: {booksdf.num_genres.max()}")
print(f"Avg number of genres: {booksdf.num_genres.mean()}")
print(f"Min number of genres: {booksdf.num_genres.min()}")
```

```
Max number of genres: 12
Avg number of genres: 5.760416666666667
Min number of genres: 1
```

In [21]:
```
flattenedgenres=[g for sub in booksdf.genre.to_list() for g in sub]
uniquegenres=list(set(flattenedgenres))
```

In [22]:
```
print(uniquegenres)
print(f"Number of unique genres: {len(uniquegenres)}") # 16 genres: maybe too many --> sparse
```

```
['biography', 'fantasy', 'thriller', 'youngadult', 'poetry', 'crime', 'graphic', 'children', 'nonfiction', 'comics', 'historicalfiction', 'history', 'mystery', 'fiction', 'romanc
e', 'paranormal']
Number of unique genres: 16
```

In [23]:
```
# encode genres
booksdf=booksdf.join(booksdf.genre.str.join('|').str.get_dummies())
```

In [24]:
```
genressum=booksdf[uniquegenres].sum()
genressum
```

Out[24]:
```
biography           23
fantasy             60
thriller            28
youngadult          66
poetry               5
crime               28
graphic             14
children            20
nonfiction          16
comics              14
historicalfiction   23
history             23
mystery             28
fiction             92
romance             53
paranormal          60
dtype: int64
```

In [25]:
```
print(f"Max amount of books in genre: {genressum.max()}") # fiction, with 96 books total and 16 nonfiction? are there fiction/nonfiction books?
print(f"Avg amount of books in genre: {genressum.mean()}")
print(f"Min amount of books in genre: {genressum.min()}") # poetry only has 5 books, could be underrepresented
```

```
Max amount of books in genre: 92
Avg amount of books in genre: 34.5625
Min amount of books in genre: 5
```

In [26]:
```
contradictorygenresdf=booksdf[(booksdf['nonfiction']==1) & (booksdf['fiction']==1)]
```

In [27]:
```
print(f"Number of books both fiction and nonfiction: {len(contradictorygenresdf)}") # 12, consider removing fiction-genre?
```

```
Number of books both fiction and nonfiction: 12
```

Join books_df (reviews) with book_metadata (books)

In [28]:
```
joinedouterdf=pd.merge(reviewdf,booksdf,left_on='book_id_mapping',right_on='book_id_mapping',how='outer')
joinedinnerdf=pd.merge(reviewdf,booksdf,left_on='book_id_mapping',right_on='book_id_mapping',how='inner')

tempdupesdf=pd.concat([joinedouterdf[['book_id_mapping','user_id_mapping']],joinedinnerdf[['book_id_mapping','user_id_mapping']]],).drop_duplicates(keep=False) # get non-matches,
differencedf=joinedouterdf.drop(index=tempdupesdf.index)

print(len(joinedouterdf))
print(len(joinedinnerdf))
len(joinedouterdf)-len(joinedinnerdf) # check number of non matches

print(len(differencedf))
print(len(tempdupesdf))
len(tempdupesdf)+len(differencedf) # checked with above so lengths match

tempcheckdf=joinedinnerdf.duplicated(tempdupesdf,False)
tempcheckdf.value_counts() # only False --> no dupes, code worked correctly
```

```
196297
19824
19824
176473
```

Out[28]:
```
False    19824
Name: count, dtype: int64
```

save new dataframes to csv files

In [29]:
```
processeddatapath=datadirpath.joinpath('processed')
try:
    processeddatapath.mkdir(exist_ok=False)
except FileExistsError:
    print('Directory already exists')

outerjoinpath=processeddatapath.joinpath('outerJoinData.csv')
innerjoinpath=processeddatapath.joinpath('innerJoinData.csv')
differencejoinpath=processeddatapath.joinpath('differenceJoinData.csv')
dupesjoinpath=processeddatapath.joinpath('duplicatesJoinData.csv')
booksdfpath=rawdatapath.joinpath('collaborative_book_metadata_with_genredummies.csv')
```

```
Directory already exists
```

In [30]:
```
joinedouterdf.to_csv(outerjoinpath,sep=';',index=False)
joinedinnerdf.to_csv(innerjoinpath,sep=';',index=False)
differencedf.to_csv(differencejoinpath,sep=';',index=False)
tempdupesdf.to_csv(dupesjoinpath,sep=';',index=False)
booksdf.to_csv(booksdfpath,sep=';',index=False)
```

Clustering

- Clustering the users
- Topic Modeling of the book blurbs
- Insights and findings

In [31]:
```
# read data, preprocess genre list, as preparation for prediction
dataworeviewsfile=datadirpath.joinpath('raw/collaborative_book_metadata_with_genredummies.csv')
models_path=cwd.joinpath('models')

fulldatadf=pd.read_csv(dataworeviewsfile,sep=';')
```

```
# remove previously created columns by pd.get_dummies()
fulldatadf=fulldatadf.drop(['biography', 'children', 'comics', 'crime', 'fantasy', 'fiction',
       'graphic', 'historicalfiction', 'history', 'mystery', 'nonfiction',
       'paranormal', 'poetry', 'romance', 'thriller', 'youngadult'],axis=1)

# preporcess genre string to list of genres
fulldatadf['genre']=fulldatadf.genre.apply(lambda x: x.replace("'","")) # remove '
fulldatadf['genre']=fulldatadf.genre.apply(lambda x: x.replace("-","")) # revome -
fulldatadf['genre']=fulldatadf.genre.apply(lambda x: x.replace(" ","")) # remove spaces
fulldatadf['genre']=fulldatadf.genre.apply(lambda x: x[1:-1].split(',')) # split into list
```

In [32]:
```
# Train or Load MultiLabelBinarizer for One Hot Encoding of genres, as preparation for prediction
retrain_mlb=True
mlbmodelpath=models_path.joinpath('mlb_model')
if retrain_mlb:
    xtrain,xtest=train_test_split(fulldatadf['genre'],test_size=0.1,random_state=randomstate,shuffle=True)
    mlb=MultiLabelBinarizer()
    mlbmodel=mlb.fit(xtrain)
    joblib.dump(mlbmodel,mlbmodelpath.joinpath('mlbmodel.pkl').as_posix())
else:
    mlbmodel=joblib.load(mlbmodelpath.joinpath('mlbmodel.pkl').as_posix())

newcols=mlbmodel.classes_

pred=mlbmodel.transform(fulldatadf.genre)
preddf=pd.DataFrame(pred,columns=newcols)
fulldatadf=fulldatadf.join(preddf)
```

User data

In [33]:
```
#Heidi Gehring: Loading user ratings and inspecting the data
user_ratings_df=pd.read_csv(rawdatapath.joinpath("collaborative_books_df.csv"))
user_ratings_df=user_ratings_df.drop([user_ratings_df.columns[0]],axis=1) # removed unnamed index column
print(user_ratings_df.columns)
print(user_ratings_df.head())
```

```
Index(['title', 'book_id', 'user_id_mapping', 'book_id_mapping',
       'Predicted Rating', 'Actual Rating'],
      dtype='object')
                title  book_id  user_id_mapping  book_id_mapping  \
0  I Am the Messenger    19057             1537              299
1  I Am the Messenger    19057            23039              299
2  I Am the Messenger    19057            39096              299
3  I Am the Messenger    19057            14631              299
4  I Am the Messenger    19057            32816              299

   Predicted Rating  Actual Rating
0               4.5              5
1               4.9              3
2               3.9              3
3               4.7              4
4               4.3              5
```

In [34]:
```
# Heidi Gehring: Count unique entries in 'user_id_mapping' to understand how many unique users there are
unique_count = user_ratings_df['user_id_mapping'].nunique()

print(f"Number of unique entries: {unique_count}")
```

```
Number of unique entries: 66909
```

In [35]:
```
#Heidi Gehring: Loading and inspecting the metadata about books
book_metadata_df=pd.read_csv(rawdatapath.joinpath("collaborative_book_metadata.csv"))
book_metadata_df=book_metadata_df.drop([book_metadata_df.columns[0]],axis=1) # removed unnamed index column
print(book_metadata_df.columns)
print(book_metadata_df.head())
```

```
Index(['book_id', 'title', 'image_url', 'url', 'num_pages', 'ratings_count',
       'description', 'genre', 'name', 'book_id_mapping'],
      dtype='object')
    book_id                                              title  \
0   5899779  Pride and Prejudice and Zombies Pride and Prej...
1    872333                          Blue Bloods Blue Bloods 1
2  15507958                      Me Before You Me Before You 1
3     66559                                   Sharp Objects
4   7235533          The Way of Kings The Stormlight Archive 1

                                           image_url  \
0  https://images.gr-assets.com/books/1320449653m...
1  https://images.gr-assets.com/books/1322281515m...
2  https://images.gr-assets.com/books/1357108762m...
3  https://images.gr-assets.com/books/1423241485m...
4  https://images.gr-assets.com/books/1507307887m...

                                                 url  num_pages  \
0  https://www.goodreads.com/book/show/5899779-pr...        320
1  https://www.goodreads.com/book/show/872333.Blu...        302
2  https://www.goodreads.com/book/show/15507958-m...        369
3  https://www.goodreads.com/book/show/66559.Shar...        254
4  https://www.goodreads.com/book/show/7235533-th...       1007

   ratings_count                                        description  \
0         105537  The New York Times Best Seller is now a major ...
1         117633  When the Mayflower set sail in 1620, it carrie...
2         609327  Louisa Clark is an ordinary young woman living...
3         208394  Fresh from a brief stay at a psych hospital, r...
4         151473  Speak again the ancient oaths,\nLife before de...

                                               genre                name  \
0  ['fantasy, paranormal', 'romance', 'fiction', ...         Jane Austen
1  ['young-adult', 'fantasy, paranormal', 'romanc...  Melissa de la Cruz
2                          ['romance', 'fiction']          Jojo Moyes
3           ['mystery, thriller, crime', 'fiction']       Gillian Flynn
4               ['fantasy, paranormal', 'fiction']    Brandon Sanderson

   book_id_mapping
0              808
1              217
2              385
3              192
4              873
```

Data preparation:

Merge the 2 datasets and only keep the user ratings, where we have book metadata.

Going forward the merged dataset will be explored to understand whether it is suitbable to predict how users would rate books.

In [36]:
```
#Heidi Gehring: prepare data about books to be merged

# Function to clean up genre names
def clean_genre(genre):
    if pd.isna(genre):
        return ''
    genre = genre.strip()
    genre = re.sub(r"[^\w\s,]", "", genre)  # Remove any non-alphanumeric characters except commas and spaces
    genre = genre.replace("'", "")          # Remove single quotes
    genre = genre.replace("[", "")          # Remove square brackets
    genre = genre.replace("]", "")
    genre = genre.replace(",", ", ")        # Normalize spaces around commas
    return genre

# Apply the cleaning function to the genre column
book_metadata_df['cleaned_genre'] = book_metadata_df['genre'].apply(clean_genre)

# Display cleaned genres to verify
print("Cleaned Genres:")
print(book_metadata_df['cleaned_genre'].unique())
```

```
Cleaned Genres:
['fantasy, paranormal, romance, fiction, history, historical fiction, biography, youngadult, mystery, thriller, crime'
 'youngadult, fantasy, paranormal, romance, fiction, mystery, thriller, crime'
 'romance, fiction' 'mystery, thriller, crime, fiction'
 'fantasy, paranormal, fiction'
 'fiction, mystery, thriller, crime, fantasy, paranormal'
 'fantasy, paranormal, fiction, history, historical fiction, biography'
 'fantasy, paranormal, fiction, youngadult'
 'fantasy, paranormal, fiction, youngadult, romance'
 'fiction, romance' 'fiction, fantasy, paranormal, youngadult'
 'fiction, poetry, history, historical fiction, biography, nonfiction, romance, youngadult'
 'fantasy, paranormal, fiction, romance, mystery, thriller, crime, youngadult'
 'fantasy, paranormal, youngadult, fiction, children, romance, mystery, thriller, crime'
 'fantasy, paranormal, fiction, youngadult, children, mystery, thriller, crime'
 'youngadult, fiction, mystery, thriller, crime, romance, fantasy, paranormal'
 'fiction, fantasy, paranormal, mystery, thriller, crime'
 'youngadult, fiction, children, fantasy, paranormal, romance'
 'youngadult, romance, fiction'
 'youngadult, fantasy, paranormal, fiction, romance'
 'youngadult, fantasy, paranormal, romance, fiction'
 'romance, youngadult, fiction'
 'fiction, fantasy, paranormal, mystery, thriller, crime, poetry, youngadult, history, historical fiction, biography'
 'fiction, youngadult, fantasy, paranormal, children'
 'fiction, romance, youngadult'
 'nonfiction, history, historical fiction, biography, fiction'
 'fantasy, paranormal, youngadult, romance, fiction'
 'fiction, fantasy, paranormal, youngadult, children'
 'poetry, children, fiction, youngadult, fantasy, paranormal, nonfiction'
 'children, fiction, fantasy, paranormal, youngadult' 'nonfiction'
 'fiction, nonfiction, youngadult, romance'
 'fantasy, paranormal, fiction, comics, graphic, youngadult'
 'romance, fiction, youngadult, fantasy, paranormal'
 'fantasy, paranormal, youngadult, fiction, romance, mystery, thriller, crime, history, historical fiction, biography'
 'fantasy, paranormal, romance, fiction'
 'youngadult, fantasy, paranormal, fiction, children, romance'
 'fantasy, paranormal, youngadult, fiction, children, romance'
 'youngadult, mystery, thriller, crime, fiction, romance'
 'fantasy, paranormal, fiction, youngadult, children'
 'fiction, history, historical fiction, biography, romance, nonfiction'
 'nonfiction, history, historical fiction, biography, youngadult, comics, graphic'
 'fantasy, paranormal, youngadult, romance, fiction, history, historical fiction, biography, mystery, thriller, crime'
 'fantasy, paranormal, children, fiction, youngadult'
 'youngadult, fiction, fantasy, paranormal, romance'
 'youngadult, fiction, children'
 'fiction, mystery, thriller, crime, history, historical fiction, biography'
 'comics, graphic, fantasy, paranormal, fiction, romance'
 'fiction, history, historical fiction, biography, romance'
 'nonfiction, history, historical fiction, biography, comics, graphic'
 'comics, graphic, fiction, fantasy, paranormal, mystery, thriller, crime'
 'romance, fiction, youngadult'
 'fiction, history, historical fiction, biography'
 'fantasy, paranormal, romance, mystery, thriller, crime, fiction, youngadult'
 'fantasy, paranormal, romance, youngadult, fiction'
 'comics, graphic, fiction, fantasy, paranormal, mystery, thriller, crime, history, historical fiction, biography'
 'romance, fiction, youngadult, poetry'
 'nonfiction, history, historical fiction, biography'
 'mystery, thriller, crime, fiction, history, historical fiction, biography'
 'youngadult, fiction, romance, children' 'nonfiction, fiction'
 'nonfiction, history, historical fiction, biography, fiction, youngadult, romance, comics, graphic'
 'nonfiction, history, historical fiction, biography, youngadult, fiction, comics, graphic, children, mystery, thriller, crime'
 'fiction, poetry, history, historical fiction, biography, fantasy, paranormal'
 'fiction, romance, youngadult, nonfiction'
 'comics, graphic, history, historical fiction, biography, nonfiction, fiction, youngadult'
 'nonfiction, history, historical fiction, biography, comics, graphic, fiction'
 'comics, graphic, nonfiction, history, historical fiction, biography, youngadult, fiction'
 'youngadult, fantasy, paranormal, fiction, children, mystery, thriller, crime']
```

Use one hot encoding to be able to use the genre information for clustering and prediction, going forward.

In [37]:
```
# One-hot encoding the genre information
genre_dummies = book_metadata_df['cleaned_genre'].str.get_dummies(sep=', ')

# Combine the encoded genre information with the book metadata
book_metadata_df = pd.concat([book_metadata_df, genre_dummies], axis=1)

# Drop the original genre and cleaned_genre columns
book_metadata_df = book_metadata_df.drop(columns=['genre', 'cleaned_genre'])


print("Processed Book Metadata DataFrame Head:")
print(book_metadata_df.head())
```

```
Processed Book Metadata DataFrame Head:
    book_id                                              title  \
0   5899779  Pride and Prejudice and Zombies Pride and Prej...
1    872333                          Blue Bloods Blue Bloods 1
2  15507958                     Me Before You Me Before You 1
3     66559                                     Sharp Objects
4   7235533          The Way of Kings The Stormlight Archive 1

                                       image_url  \
0  https://images.gr-assets.com/books/1320449653m...
1  https://images.gr-assets.com/books/1322281515m...
2  https://images.gr-assets.com/books/1357108762m...
3  https://images.gr-assets.com/books/1423241485m...
4  https://images.gr-assets.com/books/1507307887m...

                                           url  num_pages  \
0  https://www.goodreads.com/book/show/5899779-pr...        320
1  https://www.goodreads.com/book/show/872333.Blu...        302
2  https://www.goodreads.com/book/show/15507958-m...        369
3  https://www.goodreads.com/book/show/66559.Shar...        254
4  https://www.goodreads.com/book/show/7235533-th...       1007

   ratings_count                                        description  \
0         105537  The New York Times Best Seller is now a major ...
1         117633  When the Mayflower set sail in 1620, it carrie...
2         609327  Louisa Clark is an ordinary young woman living...
3         208394  Fresh from a brief stay at a psych hospital, r...
4         151473  Speak again the ancient oaths,\nLife before de...

                name  book_id_mapping   biography  ...   youngadult  \
0        Jane Austen              808           1  ...            1
1  Melissa de la Cruz              217           0  ...            0
2         Jojo Moyes              385           0  ...            0
3      Gillian Flynn              192           0  ...            0
4   Brandon Sanderson              873           0  ...            0

   children  comics  fantasy  fiction  mystery  nonfiction  poetry  romance  \
0         0       0        1        0        0           0       0        0
1         0       0        0        0        0           0       0        0
2         0       0        0        0        0           0       0        1
3         0       0        0        0        1           0       0        0
4         0       0        1        0        0           0       0        0

   youngadult
0           0
1           1
2           0
3           0
4           0

[5 rows x 34 columns]
```

Heidi Gehring: regularly use print to inspect the current dataframe

In [38]:
```
print(book_metadata_df.columns)
```

```
Index(['book_id', 'title', 'image_url', 'url', 'num_pages', 'ratings_count',
       'description', 'name', 'book_id_mapping', ' biography', ' children',
       ' comics', ' crime', ' fantasy', ' fiction', ' graphic',
       ' historical fiction', ' history', ' mystery', ' nonfiction',
       ' paranormal', ' poetry', ' romance', ' thriller', ' youngadult',
       'children', 'comics', 'fantasy', 'fiction', 'mystery', 'nonfiction',
       'poetry', 'romance', 'youngadult'],
      dtype='object')
```

In [39]:
```
#Heidi Gehring: Merge the user ratings DataFrame with the book metadata DataFrame
# Use 'book_id_mapping' as the common key, and only keep rows with metadata

# Merge the DataFrames
merged_df = pd.merge(user_ratings_df, book_metadata_df, on='book_id_mapping', how='inner')

# The 'inner' join ensures that only rows with matching 'book_id_mapping' in both DataFrames are kept.

# Display the DataFrame to verify
print("Merged and Filtered DataFrame Head:")
print(merged_df.head())
```

```
Merged and Filtered DataFrame Head:
                     title_x  book_id_x  user_id_mapping  book_id_mapping  \
0  We Should All Be Feminists   22738563            64847              873
1  We Should All Be Feminists   22738563            45548              873
2  We Should All Be Feminists   22738563             9063              873
3  We Should All Be Feminists   22738563            27200              873
4  We Should All Be Feminists   22738563            41888              873

   Predicted Rating  Actual Rating  book_id_y  \
0               4.0              5    7235533
1               4.1              4    7235533
2               4.5              5    7235533
3               4.2              5    7235533
4               4.4              5    7235533

                                  title_y  \
0  The Way of Kings The Stormlight Archive 1
1  The Way of Kings The Stormlight Archive 1
2  The Way of Kings The Stormlight Archive 1
3  The Way of Kings The Stormlight Archive 1
4  The Way of Kings The Stormlight Archive 1

                                       image_url  \
0  https://images.gr-assets.com/books/1507307887m...
1  https://images.gr-assets.com/books/1507307887m...
2  https://images.gr-assets.com/books/1507307887m...
3  https://images.gr-assets.com/books/1507307887m...
4  https://images.gr-assets.com/books/1507307887m...

                                             url  ...  youngadult  \
0  https://www.goodreads.com/book/show/7235533-th...  ...           0
1  https://www.goodreads.com/book/show/7235533-th...  ...           0
2  https://www.goodreads.com/book/show/7235533-th...  ...           0
3  https://www.goodreads.com/book/show/7235533-th...  ...           0
4  https://www.goodreads.com/book/show/7235533-th...  ...           0

   children comics fantasy  fiction  mystery  nonfiction  poetry  romance  \
0         0      0       1        0        0           0       0        0
1         0      0       1        0        0           0       0        0
2         0      0       1        0        0           0       0        0
3         0      0       1        0        0           0       0        0
4         0      0       1        0        0           0       0        0

   youngadult
0           0
1           0
2           0
3           0
4           0

[5 rows x 39 columns]
```

In [40]:
```
print(merged_df.shape)
```

```
(19824, 39)
```

In [41]:
```
# Count unique entries
unique_count1 = merged_df['user_id_mapping'].nunique()

print(f"Number of unique entries: {unique_count1}")
```

```
Number of unique entries: 17103
```

Heidi Gehring: Objective is to create 1 vector per user. This will then be used as input to create user clusters

Creating user clusters: Intially normalized data was used, but it did not yield good results for the clusters. Then the "non-normalized" data was used. It also did not yield good results for the clusters. So it was decided to move into the prediction without clustering the users.

In [42]:
```
#Heidi Gehring: reduce the data to 1 vector per user for further processing

# Step 1: Create the user-item matrix combined with book metadata
user_item_with_metadata = merged_df.copy()

# Identify numeric columns (metadata columns plus genres)
numeric_columns = ['num_pages', 'ratings_count'] + [col for col in genre_dummies.columns if pd.api.types.is_numeric_dtype(genre_dummies[col])]

# Step 2: Multiply each user's rating by the corresponding numeric metadata column
# Filter out non-numeric columns from `user_item_with_metadata`
numeric_metadata_columns = [col for col in numeric_columns if col in user_item_with_metadata.columns]

# Ensure 'Actual Rating' is numeric
user_item_with_metadata['Actual Rating'] = pd.to_numeric(user_item_with_metadata['Actual Rating'], errors='coerce')

# Apply multiplication only on numeric columns
for col in numeric_metadata_columns:
    if col in user_item_with_metadata.columns:
        user_item_with_metadata[col] = user_item_with_metadata['Actual Rating'] * user_item_with_metadata[col]

# Drop non-numeric columns before aggregation
user_item_with_metadata_numeric = user_item_with_metadata[numeric_metadata_columns + ['user_id_mapping']]

# Step 3: Aggregate the vectors for each user
user_vectors = user_item_with_metadata_numeric.groupby('user_id_mapping').mean()

# Step 4: Normalize the vectors, was initially executed, and in the second iteration not.
#user_vectors_normalized = pd.DataFrame(
 #   normalize(user_vectors, norm='l2'),
  #  index=user_vectors.index,
   # columns=user_vectors.columns
#)

# Step 5: Review the final user vectors
#print("Aggregated and Normalized User Vectors Head:") # not needed here as it was decided to try without normalization
#print(user_vectors_normalized.head())
print(user_vectors.head())
```

```
                num_pages  ratings_count   biography   children   comics  \
user_id_mapping
2                  1240.0       826584.0         0.0        0.0      0.0
4                   720.0       414009.0         0.0        3.0      0.0
7                  1164.0      1494384.0         4.0        0.0      4.0
21                  448.0       402392.0         0.0        0.0      0.0
23                  720.0       292370.0         0.0        0.0      0.0

                 crime   fantasy   fiction   graphic   historical fiction  \
user_id_mapping
2                  0.0       4.0       4.0       0.0                  0.0
4                  0.0       0.0       3.0       0.0                  0.0
7                  0.0       0.0       4.0       4.0                  4.0
21                 2.0       2.0       0.0       0.0                  0.0
23                 0.0       5.0       5.0       5.0                  0.0

                ...   youngadult  children  comics  fantasy  fiction  \
user_id_mapping  ...
2               ...          4.0       0.0     0.0      0.0      0.0
4               ...          3.0       0.0     0.0      3.0      0.0
7               ...          4.0       0.0     0.0      0.0      0.0
21              ...          0.0       0.0     0.0      0.0      2.0
23              ...          0.0       0.0     5.0      0.0      0.0

                mystery  nonfiction  poetry  romance  youngadult
user_id_mapping
2                   0.0         0.0     0.0      4.0         0.0
4                   0.0         0.0     0.0      0.0         0.0
7                   0.0         4.0     0.0      0.0         0.0
21                  0.0         0.0     0.0      0.0         0.0
23                  0.0         0.0     0.0      0.0         0.0

[5 rows x 27 columns]
```

In [43]:
```
#inspect the shape of the newly created user vectors
print(user_vectors.shape)
```

```
(17103, 27)
```

Clustering the users

The hypothesis was, that by creating clusters, it might be possible to then use the information for each cluster to predict user ratings. The result was, that no good clustering was possible.

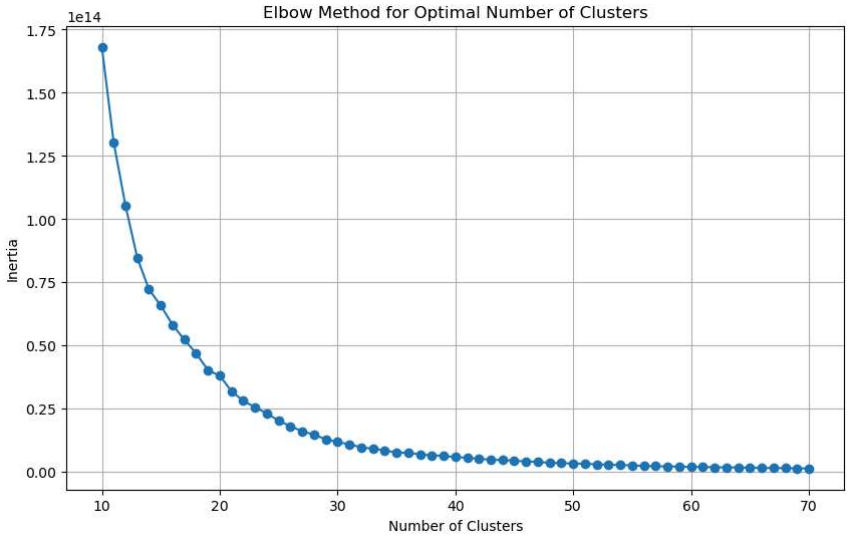
Some iterations in the code were tried out with the cluster range to come to a conclusion how many cllusters might be optimal.

According to the elbow method and the Silhouette method, a cluster number of 25 might already give an acceptable result, but expereimenting shows that neither with 25 nor with 75 clusters, satisfying results can be achieved.

In [44]:
```
#Heidi Gehring: Define the range for the number of clusters
cluster_range = range(10, 71)  # Adjust this range if needed, was used to run different variants for the clusters
```

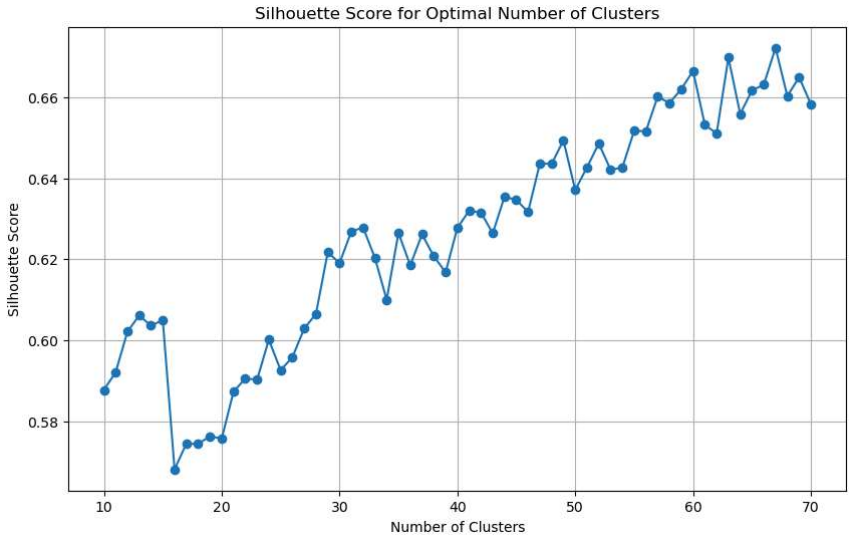
Experimenting with 20, 30 and 50 cluster range, 20 was very inconclusive, 30 was better Looking at a range up to 50 or 75 leads to the conclusion that 25 clusters could be a good number

In [45]:
```
# elbow method to find out the right number of clusters.

# Compute K-Means for each number of clusters specified in cluster_range
inertia = []
for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    kmeans.fit(user_vectors)
    inertia.append(kmeans.inertia_)

# Plot the Elbow curve
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.grid(True)
plt.show()
```



In [46]:
```
# silhouette score as alternative method to look at potential number of clusters

silhouette_scores = []
for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    cluster_labels = kmeans.fit_predict(user_vectors)
    silhouette_avg = silhouette_score(user_vectors, cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Plot the Silhouette scores
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, silhouette_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Optimal Number of Clusters')
```

```
plt.grid(True)
plt.show()
```



Running several versions of elbow and silhouette method with different cluster ranges (up to 70 clusters) leads to the initial decision to go forward with 25 clusters, as it seems to be a reasonable compromise of acceptable scores and a manageable number of clusters.

Plotting the 25 clusters shows, that the clusters are not well separated.

Trying out different numbers of clusters leads to the conclusion, that with up to 70 clusters, no good separation can be achieved.

Both 2D and 3D Visualization were applied to inspect the outcomes. No good outcomes were achieved in any visualization.

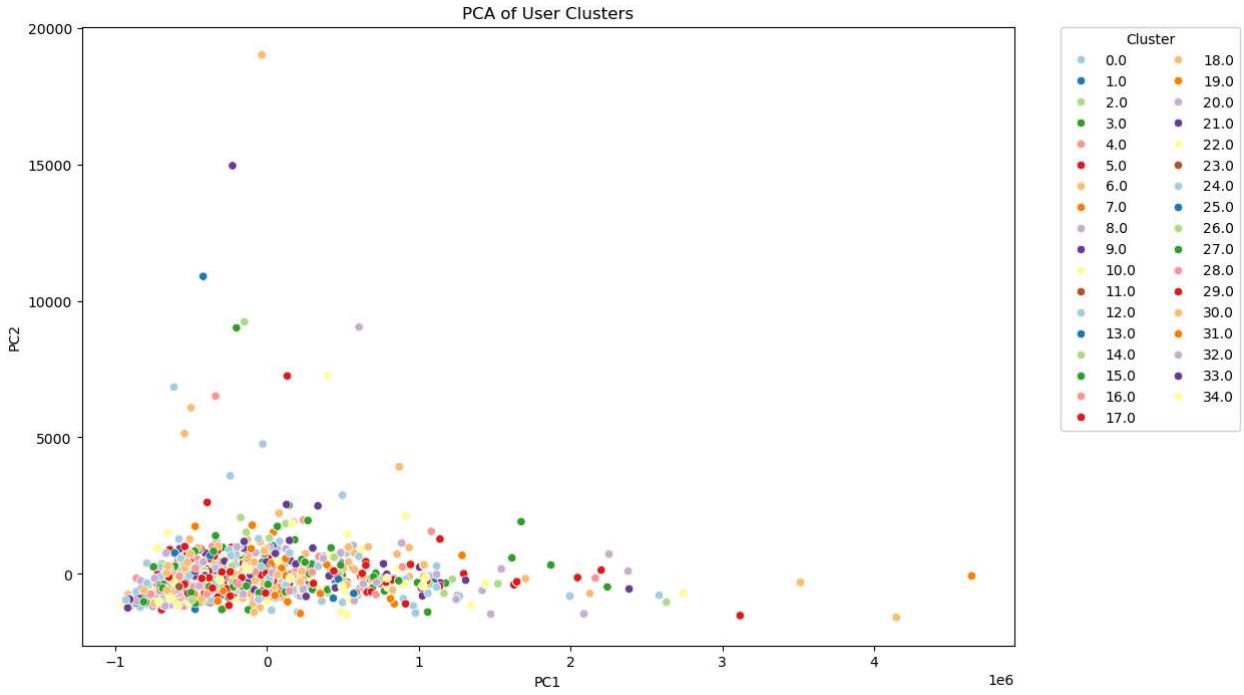
In [47]:
```
# setting the number of clusters to be calculated, can be changed to experiement with different outcomes
optimal_n_clusters = 35

# Fit K-Means model with the chosen number of clusters
kmeans = KMeans(n_clusters=optimal_n_clusters, random_state=0)
user_vectors['cluster'] = kmeans.fit_predict(user_vectors)
```

In [48]:
```
# Get cluster profiles
cluster_profiles = user_vectors.groupby('cluster').mean()
print(cluster_profiles.head())
```

```
          num_pages  ratings_count   biography   children    comics  \
cluster
0         729.698353   2.019763e+05    0.193862   0.036677  0.111527
1        2157.882883   2.867410e+06    4.331081   0.218468  0.000000
2        1756.782263   1.059470e+06    0.622730   0.550908  0.116407
3        2356.324266   2.103519e+06    2.000864   0.049223  0.033679
4        2008.033333   4.511307e+06    0.000000   3.966667  0.000000

            crime   fantasy   fiction   graphic   historical fiction  ...  \
cluster                                                               ...
0        1.014970  0.990269  2.440120  0.687874            0.193862  ...
1        0.067568  0.144144  0.425676  0.000000            4.331081  ...
2        1.129864  1.641537  3.739462  0.159209            0.622730  ...
3        0.076857  0.084629  4.319516  0.045337            2.000864  ...
4        0.000000  0.000000  3.966667  0.000000            0.000000  ...

          youngadult  children    comics   fantasy   fiction   mystery  \
cluster
0           1.301647  0.000000  0.576347  1.062874  0.049401  0.014970
1           0.385135  0.033784  0.000000  0.202703  4.457207  0.027027
2           2.336089  0.444066  0.042802  1.136835  0.416991  0.238327
3           2.424007  0.031088  0.011658  2.571244  0.174439  0.005181
4           3.966667  0.000000  0.000000  3.966667  0.044444  0.000000

         nonfiction    poetry   romance  youngadult
cluster
0          0.366766  0.000000  0.251497    0.441617
1          0.051802  0.076577  0.000000    0.033784
2          0.218547  0.003891  0.648833    1.013457
3          1.665371  0.000000  0.015544    0.019430
4          0.000000  0.000000  0.000000    0.000000

[5 rows x 27 columns]
```

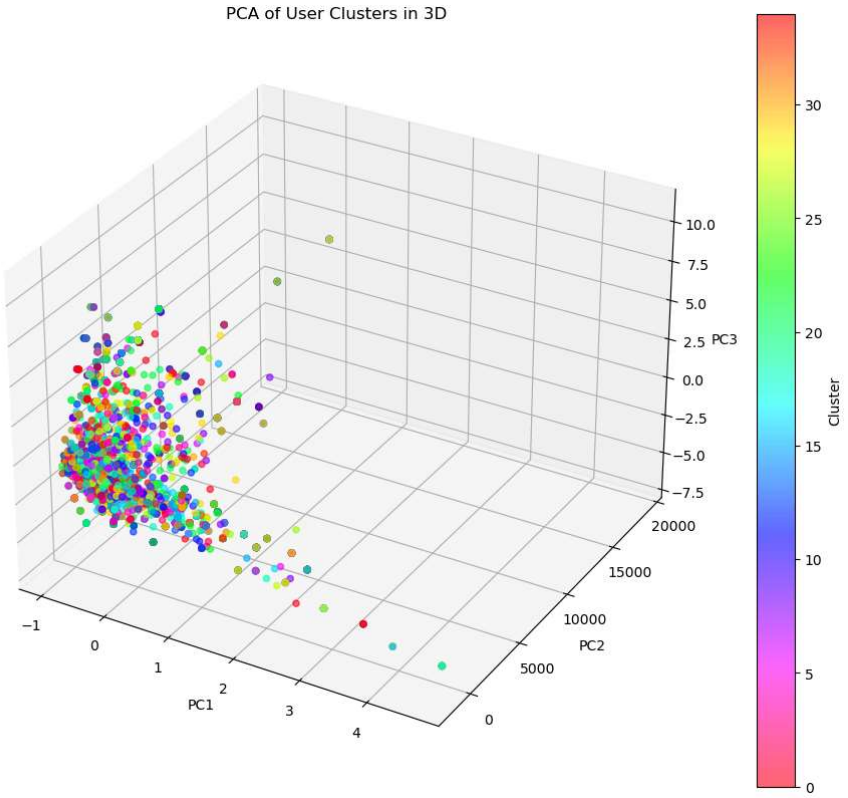
Initially the viridis color palette was used, but this is very monotonic. A change to Paired helped the visualization in general, but not the outcome.

In [49]:
```
# Reduce to 2D for visualization
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(user_vectors.drop('cluster', axis=1))

# Create a DataFrame for plotting
df_pca = pd.DataFrame(reduced_data, columns=['PC1', 'PC2'])
df_pca['cluster'] = user_vectors['cluster']

# Plot the clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='cluster', palette='Paired', legend='full')
plt.title('PCA of User Clusters')
# Adjust the legend to fit into the visual
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0, ncol=2)
plt.show()
```



In [50]:
```
#Creating 3d visualization, to see if this leads to a better visualization
# # prepare data for it

# Perform PCA to reduce to 3D
pca = PCA(n_components=3)
reduced_data = pca.fit_transform(user_vectors.drop('cluster', axis=1))

# Create a DataFrame for plotting
df_pca = pd.DataFrame(reduced_data, columns=['PC1', 'PC2', 'PC3'])
df_pca['cluster'] = user_vectors['cluster']
```

In [51]:
```
#create 3D scatter plot, starting with Viridis again

# Create a 3D scatter plot
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
scatter = ax.scatter(df_pca['PC1'], df_pca['PC2'], df_pca['PC3'],
                     c=df_pca['cluster'], cmap='hsv_r', alpha=0.6)

# Add color bar
cbar = plt.colorbar(scatter)
cbar.set_label('Cluster')

# Labels and title
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.set_title('PCA of User Clusters in 3D')

plt.show()
```



Clustering with DBSCAN

As the approach using K-Means did not yield statisfying results, another approach was tried, using the DB Scan method. In general, the same steps as for the first clusterin appraoch were followed.

In [52]:
```
# Following the same steps as for K-Means Clustering, steps are repeated as normalization in step 4 is used here
# Step 1: Create the user-item matrix combined with book metadata
user_item_with_metadata = merged_df.copy()

# Identify numeric columns (metadata columns plus genre dummies)
numeric_columns = ['num_pages', 'ratings_count'] + [col for col in genre_dummies.columns if pd.api.types.is_numeric_dtype(genre_dummies[col])]

# Step 2: Multiply each user's rating by the corresponding numeric metadata column
# Filter out non-numeric columns from `user_item_with_metadata`
numeric_metadata_columns = [col for col in numeric_columns if col in user_item_with_metadata.columns]

# Ensure 'Actual Rating' is numeric
user_item_with_metadata['Actual Rating'] = pd.to_numeric(user_item_with_metadata['Actual Rating'], errors='coerce')

# Apply multiplication only on numeric columns
for col in numeric_metadata_columns:
    if col in user_item_with_metadata.columns:
        user_item_with_metadata[col] = user_item_with_metadata['Actual Rating'] * user_item_with_metadata[col]

# Drop non-numeric columns before aggregation
user_item_with_metadata_numeric = user_item_with_metadata[numeric_metadata_columns + ['user_id_mapping']]

# Step 3: Aggregate the vectors for each user
user_vectors = user_item_with_metadata_numeric.groupby('user_id_mapping').mean()

# Step 4: Normalization
user_vectors_normalized = pd.DataFrame(
   normalize(user_vectors, norm='l2'),
   index=user_vectors.index,
   columns=user_vectors.columns
)

# Step 5: Review the final user vectors
print("Aggregated and Normalized User Vectors Head:")
print(user_vectors_normalized.head())
```

```
Aggregated and Normalized User Vectors Head:
                num_pages  ratings_count   biography   children    comics  \
user_id_mapping
2                0.001500       0.999999    0.000000   0.000000  0.000000
4                0.001739       0.999998    0.000000   0.000007  0.000000
7                0.000779       1.000000    0.000003   0.000000  0.000003
21               0.001113       0.999999    0.000000   0.000000  0.000000
23               0.002463       0.999997    0.000000   0.000000  0.000000

                   crime   fantasy   fiction   graphic   historical fiction  \
user_id_mapping
2                0.000000  0.000005  0.000005  0.000000             0.000000
4                0.000000  0.000000  0.000007  0.000000             0.000000
7                0.000000  0.000000  0.000003  0.000003             0.000003
21               0.000005  0.000005  0.000000  0.000000             0.000000
23               0.000000  0.000017  0.000017  0.000017             0.000000

                 ...   youngadult  children    comics   fantasy   fiction  \
user_id_mapping  ...
2                ...     0.000005       0.0  0.000000  0.000000  0.000000
4                ...     0.000007       0.0  0.000000  0.000007  0.000000
7                ...     0.000003       0.0  0.000000  0.000000  0.000000
21               ...     0.000000       0.0  0.000000  0.000000  0.000005
23               ...     0.000000       0.0  0.000017  0.000000  0.000000

                 mystery  nonfiction  poetry   romance  youngadult
user_id_mapping
2                    0.0    0.000000     0.0  0.000005         0.0
4                    0.0    0.000000     0.0  0.000000         0.0
7                    0.0    0.000003     0.0  0.000000         0.0
21                   0.0    0.000000     0.0  0.000000         0.0
23                   0.0    0.000000     0.0  0.000000         0.0

[5 rows x 27 columns]
```

In [53]:
```
# Standardize the data (important for DBSCAN)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(user_vectors_normalized)
```

In [54]:
```
# Initialize DBSCAN with chosen parameters
dbscan = DBSCAN(eps=1, min_samples=5)

# Fit the model to the scaled data
dbscan.fit(X_scaled)

#  Labels assigned to each data point
labels = dbscan.labels_
```

In [55]:
```
# Number of clusters (excluding noise)
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

# Number of noise points
n_noise = list(labels).count(-1)

print(f'Estimated number of clusters: {n_clusters}')
print(f'Estimated number of noise points: {n_noise}')
```

```
Estimated number of clusters: 72
Estimated number of noise points: 891
```

With the parameters eps =1 und min samples = 5 a silhousett score of 0,379 was achieved, which is not satisfying.

In [56]:
```
# Calculate silhouette score (ignoring noise)
silhouette = silhouette_score(X_scaled, labels) if len(set(labels)) > 1 else -1
print(f'Silhouette Score: {silhouette}')
```

```
Silhouette Score: 0.28371653379391715
```

Running the random search cv leads to the result that the parameters eps=1 and min-samples = 2 are best. This results in >400 clusters which is also not desired. Therefore the DB Scan clusterin method will not be explored further.

In [57]:
```
# #random search cv
# dbscan = DBSCAN()

# param_dist = {
#     'eps': np.linspace(1, 5, 5),  # setting the epsilon parameter
#     'min_samples': range(2, 20)        # setting min samples to form a cluster
# }


# def ari_scorer(estimator, X, y_true):
#     cluster_labels = estimator.fit_predict(X)
#     if len(set(cluster_labels)) > 1:  # Ensure there's more than one cluster
#         return adjusted_rand_score(y_true, cluster_labels)
#     else:
#         return -1  # Assign a low score if clustering is trivial

# # Example usage in RandomizedSearchCV
# ari = make_scorer(ari_scorer, needs_proba=False)



# # RandomizedSearchCV setup
# random_search = RandomizedSearchCV(
#     estimator=dbscan,
#     param_distributions=param_dist,
#     n_iter=100,                   # Number of parameter settings to sample
#     scoring=ari,           # Use the custom silhouette scoring function
#     cv=3,                         # Number of folds (not used, but required by RandomizedSearchCV)
#     verbose=1,
#     random_state=42,
#     n_jobs=-1
# )


# random_search.fit(X_scaled)

# # Best hyperparameters and corresponding score
# print("Best Hyperparameters:", random_search.best_params_)
# print("Best Silhouette Score:", random_search.best_score_)

'''
We are commenting this code out, as the output is quite large and did not provide usable insights.
'''
```

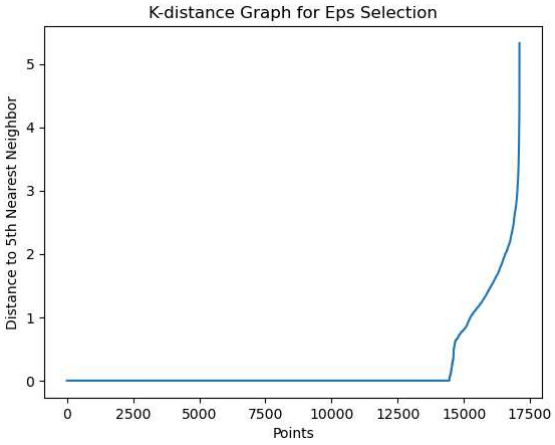
Out[57]:
```
'\nWe are commenting this code out, as the output is quite large and did not provide usable insights.\n'
```

Plotting the result verifies that this is really not helpfull in moving forward.

In [58]:
```
# Strip leading and trailing spaces from the column names
user_vectors.columns = user_vectors.columns.str.strip()

# Verify that the column names have been cleaned
print(user_vectors.columns)
```

```
Index(['num_pages', 'ratings_count', 'biography', 'children', 'comics',
       'crime', 'fantasy', 'fiction', 'graphic', 'historical fiction',
       'history', 'mystery', 'nonfiction', 'paranormal', 'poetry', 'romance',
       'thriller', 'youngadult', 'children', 'comics', 'fantasy', 'fiction',
       'mystery', 'nonfiction', 'poetry', 'romance', 'youngadult'],
      dtype='object')
```

In [59]:
```
# Assuming 'user_vectors' is your DataFrame with features
# Select only the relevant columns for clustering (e.g., drop 'cluster' or non-feature columns)
features = user_vectors[['num_pages', 'ratings_count', 'biography', 'children', 'comics',
                         'crime', 'fantasy', 'fiction', 'graphic', 'historical fiction',
                         'history', 'mystery', 'nonfiction', 'paranormal', 'poetry',
                         'romance', 'thriller', 'youngadult']]  # Example of selected columns

# Step 1: Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features)

# Step 2: Define NearestNeighbors and compute distances
neighbors = NearestNeighbors(n_neighbors=5)  # Set n_neighbors to min_samples or another reasonable value
neighbors_fit = neighbors.fit(X_scaled)

# Step 3: Find the k-neighbors for each point
distances, indices = neighbors_fit.kneighbors(X_scaled)

# Step 4: Sort the distances for k-nearest neighbors and plot
distances = np.sort(distances[:, 4], axis=0)  # 4 corresponds to the 5th nearest neighbor (index 4 = n_neighbors - 1)
plt.plot(distances)
plt.title('K-distance Graph for Eps Selection')
plt.xlabel('Points')
plt.ylabel('Distance to 5th Nearest Neighbor')
plt.show()
```



In [65]:
```
# Fit DB-Scan model
dbscan = DBSCAN(eps=0.1, min_samples=2)

user_vectors['cluster'] = dbscan.fit_predict(user_vectors)
```

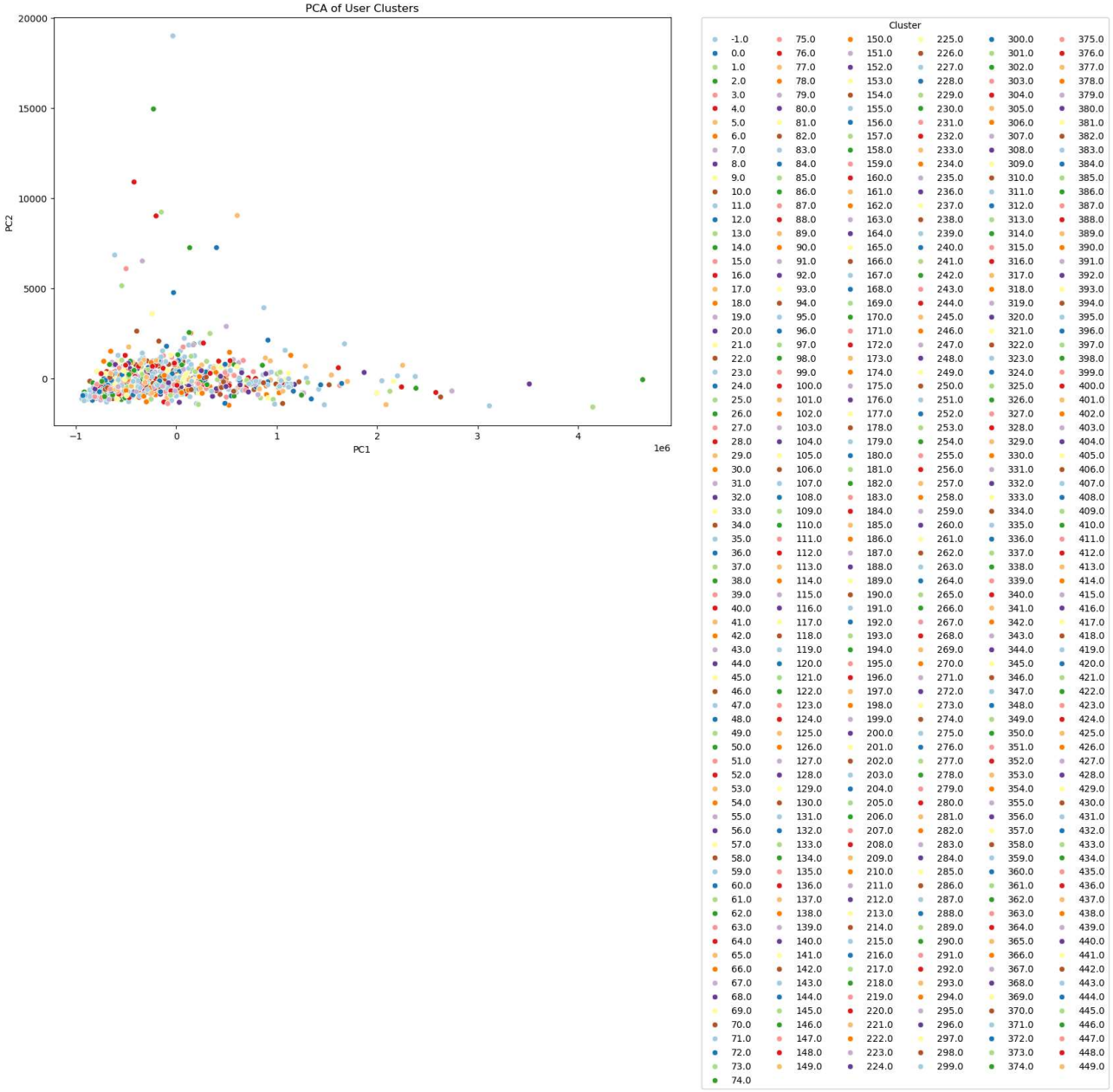
In [66]:
```
# Get cluster profiles
cluster_profiles = user_vectors.groupby('cluster').mean()
print(cluster_profiles.head())
```

```
          num_pages  ratings_count  biography  children    comics     crime  \
cluster
-1       1494.978948   9.550478e+05   0.886546  0.702839  0.285074  1.117332
 0       1240.000000   8.265840e+05   0.000000  0.000000  0.000000  0.000000
 1        720.000000   4.140890e+05   0.000000  3.000000  0.000000  0.000000
 2       1164.000000   1.494384e+06   4.000000  0.000000  4.000000  0.000000
 3        448.000000   4.023920e+05   0.000000  0.000000  0.000000  2.000000

          fantasy   fiction   graphic  historical fiction  ...  youngadult  \
cluster                                                    ...
-1       1.114097  3.042178  0.513439            0.886546  ...    2.058673
 0       4.000000  4.000000  0.000000            0.000000  ...    4.000000
 1       0.000000  3.000000  0.000000            0.000000  ...    3.000000
 2       0.000000  4.000000  4.000000            4.000000  ...    4.000000
 3       2.000000  0.000000  0.000000            0.000000  ...    0.000000

         children    comics   fantasy   fiction   mystery  nonfiction  \
cluster
-1       0.098984  0.228365  1.341245  0.666771  0.073136    0.455267
 0       0.000000  0.000000  0.000000  0.000000  0.000000    0.000000
 1       0.000000  0.000000  3.000000  0.000000  0.000000    0.000000
 2       0.000000  0.000000  0.000000  0.000000  0.000000    4.000000
 3       0.000000  0.000000  0.000000  2.000000  0.000000    0.000000

          poetry   romance  youngadult
cluster
-1       0.02906  0.237636    0.660511
 0       0.00000  4.000000    0.000000
 1       0.00000  0.000000    0.000000
 2       0.00000  0.000000    0.000000
 3       0.00000  0.000000    0.000000

[5 rows x 27 columns]
```

Plotting the clusters that the DBScan Method is creating. The result is not nice, and too many clusters are identified.

In [69]:
```
#initially it was viridis color palette, but that was very monotonic, changed it to Paired, still not very helpful

# Reduce to 2D for visualization
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(user_vectors.drop('cluster', axis=1))

# Create a DataFrame for plotting
df_pca = pd.DataFrame(reduced_data, columns=['PC1', 'PC2'])
```

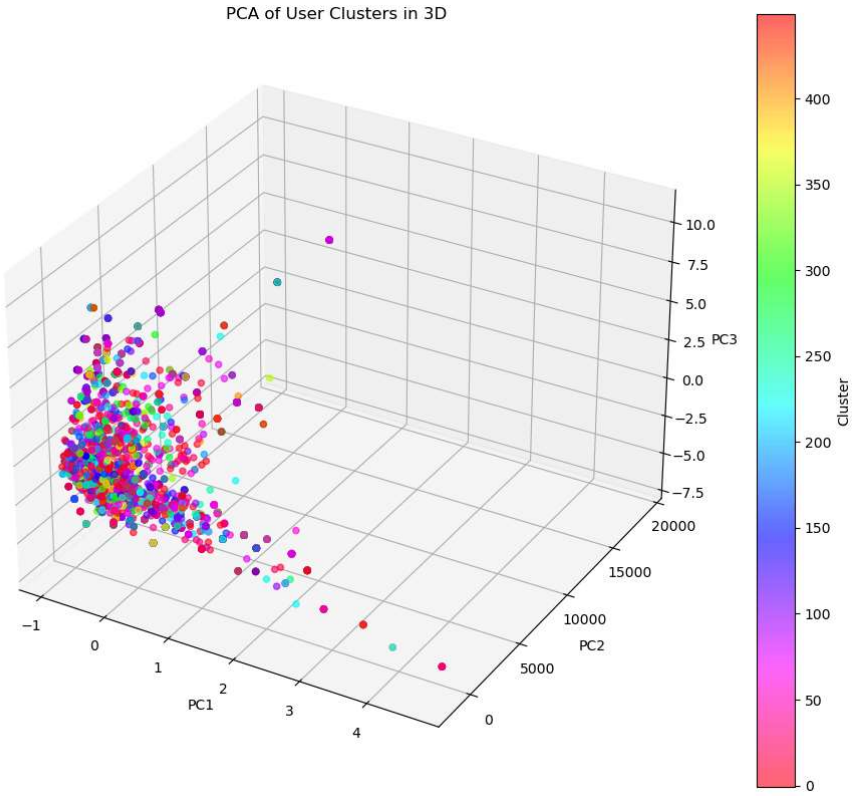
```
df_pca['cluster'] = user_vectors['cluster']

# Plot the clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='cluster', palette='Paired', legend='full')
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0, ncol=6)
plt.title('PCA of User Clusters')
plt.show()
```



Experimenting with 3D visualization to see if this helps to evaluate the clustering. It verifies that no good clusters could be created, even with a very high number of clusters.

In [71]:
```
# 3d visualization

# Perform PCA to reduce to 3D
pca = PCA(n_components=3)
reduced_data = pca.fit_transform(user_vectors.drop('cluster', axis=1))

# Create a DataFrame for plotting
df_pca = pd.DataFrame(reduced_data, columns=['PC1', 'PC2', 'PC3'])
df_pca['cluster'] = user_vectors['cluster']
```

In [72]:
```
#create 3D scatter plot, starting with Viridis again

# Create a 3D scatter plot
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
scatter = ax.scatter(df_pca['PC1'], df_pca['PC2'], df_pca['PC3'],
                     c=df_pca['cluster'], cmap='hsv_r', alpha=0.6)

# Add color bar
cbar = plt.colorbar(scatter)
cbar.set_label('Cluster')

# Labels and title
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.set_title('PCA of User Clusters in 3D')

plt.show()
```



Insights/ Findings

We started with the hypthesis, that creating clusters of users would be beneficial as input for the following work to predict how users or user within a certain cluster would rate a certain book. Trying out different ways to cluster the user data and evaluating the results shows, that no good way could be found to cluster the data to use it going forward. Therefore the decision was made to not use clusters for prediction, but use the data about the individual users and experiment with prediction based on the users directly.

Topic Modeling of book blurb

In [73]:
```
#  helper functions for preprocessing
def remove_punctuation(x):
    cleanedtext=re.sub('[^A-Za-z0-9]+', ' ', x)
    return cleanedtext

def tokenize_column(column): # this function tests gensim tokenize function used in vectorizer, not used
    l=[]
    for x in range(len(column)):
        l.append(list(tokenize(column.iloc[x])))
    return l

fulldatadf['description']=fulldatadf.description.transform(remove_punctuation)
# fulldatadf['description_list']=tokenize_column(fulldatadf.description)
# fulldatadf['description_list']
```

In [74]:
```
# xtrain,xtest=train_test_split(fulldatadf['description'],test_size=0.1,random_state=randomstate,shuffle=True)
```

group books via blurb content

ideas:

- vectorize via topics from topic modeling
- tfidf -> lda
  - evaluate via topic coherence

In [75]:
```
def topiccoherencescorer(pipe,x):
    n_top_words = 10 #higher value for higher coherence, i.e. more word to make connections for coherence
    topics=pipe.named_steps['lda'].components_
    texts=[[word for word in doc.split()] for doc in x]

    dictionary=corpora.Dictionary(texts)
    corpus=[dictionary.doc2bow(text) for text in texts]
    feature_names = [dictionary[i] for i in range(len(dictionary))]
    top_words = []
    for topic in topics:
        top_words.append([feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]])
    cm = CoherenceModel(topics=top_words, texts=texts, dictionary=dictionary, coherence='c_v') #other coherence metrics possible, sticking with c_v
    return cm.get_coherence()
```

optima study for tfidf+lda

In [76]:
```
retrain_studydescription_tfidflda=True
studyname='description_tfidflda_study'
thismodelpath=models_path.joinpath(f'{studyname}')
try:
    thismodelpath.mkdir(exist_ok=False)
except FileExistsError:
    print('Directory already exists')
if retrain_studydescription_tfidflda:
    xtrain,xtest=train_test_split(fulldatadf['description'],test_size=0.1,random_state=randomstate,shuffle=True)
    xtrain,xtest=xtrain.reset_index(drop=True),xtest.reset_index(drop=True) #hf.split() needs reset index; if drop=False, index turned into new column


    def inst_tfidf(trial:Trial)->TfidfVectorizer:
        params={
            'norm':trial.suggest_categorical('norm',['l1','l2', None]),
```

```
            'smooth_idf':trial.suggest_categorical('smooth_idf',[True,False]),
            'sublinear_tf':trial.suggest_categorical('sublinear_tf',[True,False]),
            'stop_words':trial.suggest_categorical('stop_words',['english',list(STOPWORDS)]),
            'tokenizer':tokenize,
            # 'max_df':trial.suggest_float('max_df',0,1),
            # 'min_df':trial.suggest_float('min_df',0,1),
            # 'max_features':trial.suggest_categorical('max_features',[None,100,50,25,10]) #can't use int, because of None, PS: no max features, filters out too many words
            }
        return TfidfVectorizer(**params)
    def inst_lda(trial:Trial)->LatentDirichletAllocation:
        params={
            'n_components':trial.suggest_int('n_components',3,20),
            # 'learning_method':trial.suggest_categorical('learning_method',['batch','online']),
            # 'learning_decay':trial.suggest_float('learning_decay',0.5,0.9),
            # 'learning_offset':trial.suggest_float('learning_offset',2,20),
            # 'max_iter':trial.suggest_int('max_iter',2,20),
            # 'batch_size':trial.suggest_int('batch_size',5,20),
            # 'max_doc_update_iter':trial.suggest_int('max_doc_update_iter',0,10),
            'n_jobs':-1,
            'random_state':randomstate
        }
        return LatentDirichletAllocation(**params)
    def inst_pipeTFLDA(trial:Trial)->Pipeline:
        pipeline=Pipeline([
            ('vectorizer',inst_tfidf(trial)),
            ('lda',inst_lda(trial))
        ])
        return pipeline

    def objective(trial:Trial,xtrain:pd.DataFrame)->float:
        kf = KFold(n_splits=5, shuffle=True, random_state=randomstate)
        model=inst_pipeTFLDA(trial)
        scores=[]
        for x in kf.split(xtrain):
            subxtrain=xtrain.loc[x[0]]

            pipe=model.fit(subxtrain)
            score=topiccoherencescorer(pipe, subxtrain) #prev no cv, as scorer not compatible
            scores.append(score)

        # scores = cross_val_score(model, x, cv=kf) #using default scorer --> approx log-likelohood
        # return np.min([np.mean(scores), np.median(scores)])
        return np.mean(scores)
    storage=thismodelpath.joinpath("description_tfidflda_study.db")
    if storage.exists():
        storage.unlink()
    else:
        print("No sqlite db found")

    study=create_study(study_name=studyname,direction='maximize',storage=f'sqlite:///{storage.as_posix()}',load_if_exists=False) #TPESampler used as default, no pruning
    study.optimize(lambda trial: objective(trial,xtrain),n_trials=25,n_jobs=-1,show_progress_bar=True)
    joblib.dump(study,thismodelpath.joinpath(f'study_{study.study_name}.pkl').as_posix())

    model = inst_pipeTFLDA(study.best_trial)
    fitpipe=model.fit(fulldatadf.description)
    joblib.dump(fitpipe,thismodelpath.joinpath(f'fitpipelinedefault.pkl').as_posix())

else:
    fitpipe=joblib.load(thismodelpath.joinpath(f'fitpipelinedefault.pkl').as_posix())
```

```
Directory already exists
[I 2024-09-08 15:43:01,351] A new study created in RDB with name: description_tfidflda_study
Best trial: 10. Best value: 0.624151:   4%|▍          | 1/25 [03:48<1:31:17, 228.21s/it]
[I 2024-09-08 15:46:49,532] Trial 10 finished with value: 0.6241511942136642 and parameters: {'norm': 'l1', 'smooth_idf': False, 'sublinear_tf': True, 'stop_words': 'english', 'n
_components': 3}. Best is trial 10 with value: 0.6241511942136642.
Best trial: 10. Best value: 0.624151:   4%|▍          | 1/25 [03:49<1:31:17, 228.21s/it]
[I 2024-09-08 15:46:50,847] Trial 3 finished with value: 0.6385176617787506 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': 'english', 'n_c
omponents': 9}. Best is trial 3 with value: 0.6385176617787506.
[I 2024-09-08 15:46:50,938] Trial 15 finished with value: 0.6293798566278095 and parameters: {'norm': 'l1', 'smooth_idf': False, 'sublinear_tf': True, 'stop_words': 'english', 'n
_components': 7}. Best is trial 3 with value: 0.6385176617787506.
Best trial: 3. Best value: 0.641118:  12%|█▏         | 3/25 [03:50<18:58, 51.77s/it]
[I 2024-09-08 15:46:51,861] Trial 7 finished with value: 0.6411183319539064 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': False, 'stop_words': ['my', 'bill',
'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because',
'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'nei
ther', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see',
'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own',
'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itsel
f', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'othe
r', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'ni
ne', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first',
'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein',
'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thu
s', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 't
wenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywher
e', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never',
'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however',
'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unl
ess', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'v
ery', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_compone
nts': 8}. Best is trial 7 with value: 0.6411183319539064.
Best trial: 7. Best value: 0.641118:  16%|█▌         | 4/25 [03:53<11:11, 32.00s/it]
[I 2024-09-08 15:46:54,837] Trial 9 finished with value: 0.6260434982099576 and parameters: {'norm': 'l1', 'smooth_idf': False, 'sublinear_tf': True, 'stop_words': 'english', 'n_
components': 8}. Best is trial 7 with value: 0.6411183319539064.
[I 2024-09-08 15:46:54,933] Trial 4 finished with value: 0.6212449657268124 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': 'english', 'n_c
omponents': 17}. Best is trial 7 with value: 0.6411183319539064.

[I 2024-09-08 15:46:55,053] Trial 1 finished with value: 0.6309147168070753 and parameters: {'norm': 'l2', 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': ['my', 'bil
l', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'becaus
e', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put',
'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 's
ee', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four',
'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part',
'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how',
'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'b
e', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon',
'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'th
erein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whera
s', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'th
ree', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together',
'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'n
ever', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'ho
wever', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselve
s', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'b
oth', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n
_components': 17}. Best is trial 7 with value: 0.6411183319539064.
[I 2024-09-08 15:46:55,146] Trial 5 finished with value: 0.6380689473015876 and parameters: {'norm': 'l2', 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': 'english', 'n
_components': 7}. Best is trial 7 with value: 0.6411183319539064.
```

[I 2024-09-08 15:46:55,193] Trial 8 finished with value: 0.631707610623633 and parameters: {'norm': None, 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': 'english', 'n_components': 12}. Best is trial 7 with value: 0.6411183319539064.

Best trial: 7. Best value: 0.641118: 28%|███ | 7/25 [03:54<02:51, 9.51s/it]

[I 2024-09-08 15:46:55,467] Trial 13 finished with value: 0.6180533711683664 and parameters: {'norm': None, 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 18}. Best is trial 7 with value: 0.6411183319539064.

[I 2024-09-08 15:46:55,475] Trial 0 finished with value: 0.6186761858703398 and parameters: {'norm': 'l1', 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': 'english', 'n_components': 13}. Best is trial 7 with value: 0.6411183319539064.

[I 2024-09-08 15:46:55,477] Trial 14 finished with value: 0.6372701220799267 and parameters: {'norm': 'l2', 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': 'english', 'n_components': 20}. Best is trial 7 with value: 0.6411183319539064.

[I 2024-09-08 15:46:55,481] Trial 12 finished with value: 0.6280503141198077 and parameters: {'norm': 'l2', 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': 'english', 'n_components': 15}. Best is trial 7 with value: 0.6411183319539064.

[I 2024-09-08 15:46:55,578] Trial 2 finished with value: 0.6216604085433245 and parameters: {'norm': None, 'smooth_idf': False, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 11}. Best is trial 7 with value: 0.6411183319539064.

Best trial: 7. Best value: 0.641118: 60%|█████ | 15/25 [03:54<00:19, 1.90s/it]

[I 2024-09-08 15:46:55,643] Trial 11 finished with value: 0.6201859977865471 and parameters: {'norm': None, 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 10}. Best is trial 7 with value: 0.6411183319539064.

[I 2024-09-08 15:46:55,645] Trial 6 finished with value: 0.6208578998267893 and parameters: {'norm': None, 'smooth_idf': False, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 18}. Best is trial 7 with value: 0.6411183319539064.

Best trial: 7. Best value: 0.641118: 68%|██████ | 17/25 [05:45<01:25, 10.69s/it]

[I 2024-09-08 15:48:46,598] Trial 16 finished with value: 0.6397884527312674 and parameters: {'norm': 'l2', 'smooth_idf': False, 'sublinear_tf': False, 'stop_words': 'english', 'n_components': 11}. Best is trial 7 with value: 0.6411183319539064.

Best trial: 17. Best value: 0.649728: 72%|██████ | 18/25 [05:48<01:09, 9.86s/it]

[I 2024-09-08 15:48:49,898] Trial 17 finished with value: 0.6497281726212079 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 3}. Best is trial 17 with value: 0.6497281726212079.

Best trial: 17. Best value: 0.649728: 72%|██████ | 18/25 [05:50<01:09, 9.86s/it]

[I 2024-09-08 15:48:51,439] Trial 19 finished with value: 0.6241733761934759 and parameters: {'norm': 'l2', 'smooth_idf': True, 'sublinear_tf': False, 'stop_words': 'english', 'n_components': 15}. Best is trial 17 with value: 0.6497281726212079.

Best trial: 17. Best value: 0.649728: 80%|██████ | 20/25 [05:50<00:38, 7.74s/it]

```
[I 2024-09-08 15:48:52,110] Trial 18 finished with value: 0.6497281726212079 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amougst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 3}. Best is trial 17 with value: 0.6497281726212079.
Best trial: 17. Best value: 0.649728:  80%|████████  | 20/25 [05:53<00:38,  7.74s/it]
[I 2024-09-08 15:48:54,893] Trial 21 finished with value: 0.6221220642480134 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': False, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amougst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 4}. Best is trial 17 with value: 0.6497281726212079.
Best trial: 17. Best value: 0.649728: 100%|████████| 25/25 [05:54<00:00, 14.16s/it]
[I 2024-09-08 15:48:55,318] Trial 24 finished with value: 0.6497281726212079 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amougst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 3}. Best is trial 17 with value: 0.6497281726212079.
[I 2024-09-08 15:48:55,365] Trial 23 finished with value: 0.6290793156320923 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amougst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 4}. Best is trial 17 with value: 0.6497281726212079.
[I 2024-09-08 15:48:55,371] Trial 20 finished with value: 0.6221220642480134 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': False, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amougst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 4}. Best is trial 17 with value: 0.6497281726212079.
[I 2024-09-08 15:48:55,426] Trial 22 finished with value: 0.6290793156320923 and parameters: {'norm': 'l1', 'smooth_idf': True, 'sublinear_tf': True, 'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amougst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 4}. Best is trial 17 with value: 0.6497281726212079.
```

In [77]:
```
no_top_words=10
topics=fitpipe.named_steps['lda'].components_
```

```
for topic in topics:
    print(" ".join([fitpipe.named_steps['vectorizer'].get_feature_names_out()[i] for i in topic.argsort()[:-no_top_words - 1:-1]]))
```

```
s life t way horn new time long law love
s story world new tale life love come time age
story son harry father s unexpected t youngest wanted past
```

optima study for cv+lda

In [78]:
```
retrain_studydescription_countveclda=True

studyname='description_countlda_study'
thismodelpath=models_path.joinpath(f'{studyname}')
try:
    thismodelpath.mkdir(exist_ok=False)
except FileExistsError:
    print('Directory already exists')
if retrain_studydescription_countveclda:
    xtrain,xtest=train_test_split(fulldatadf['description'],test_size=0.1,random_state=randomstate,shuffle=True)
    xtrain,xtest=xtrain.reset_index(drop=True),xtest.reset_index(drop=True) #kf.split() needs reset index; if drop=False, index turned into new column

    def inst_countvec(trial:Trial)->CountVectorizer:
        params={
            'stop_words':trial.suggest_categorical('stop_words',['english',list(STOPWORDS)]),
            'tokenizer':tokenize,
            # 'max_df':trial.suggest_float('max_df',0,1),
            # 'min_df':trial.suggest_float('min_df',0,1),
            # 'max_features':trial.suggest_categorical('max_features',[None,100,50,25,10]) #can't use int, because of None; do not set! will filter too many words
            }
        return CountVectorizer(**params)
    def inst_lda(trial:Trial)->LatentDirichletAllocation:
        params={
            'n_components':trial.suggest_int('n_components',3,20),
            # 'learning_method':trial.suggest_categorical('learning_method',['batch','online']),
            # 'learning_decay':trial.suggest_float('learning_decay',0.5,0.9),
            # 'learning_offset':trial.suggest_float('learning_offset',2,20),
            # 'max_iter':trial.suggest_int('max_iter',2,20),
            # 'batch_size':trial.suggest_int('batch_size',5,20),
            # 'max_doc_update_iter':trial.suggest_int('max_doc_update_iter',0,10),
            'n_jobs':-1,
            'random_state':randomstate
        }
        return LatentDirichletAllocation(**params)
    def inst_pipeTFLDA(trial:Trial)->Pipeline:
        pipeline=Pipeline([
            ('vectorizer',inst_countvec(trial)),
            ('lda',inst_lda(trial))
        ])
        return pipeline

    def objective(trial:Trial,x:pd.DataFrame)->float:
        kf = KFold(n_splits=5, shuffle=True, random_state=randomstate)
        model=inst_pipeTFLDA(trial)
        scores=[]
        for x in kf.split(xtrain):
            subxtrain=xtrain.loc[x[0]]

            pipe=model.fit(subxtrain)
            score=topiccoherencescorer(pipe,subxtrain) #no cv, as scorer not compatible
            scores.append(score)

            # scores = cross_val_score(model, x, cv=kf) #using default scorer --> approx log-likelohood
        # return np.min([np.mean(scores), np.median(scores)])
        return np.mean(scores)

    storage=thismodelpath.joinpath("description_countveclda_studydefault.db")
    if storage.exists():
        storage.unlink()
    else:
        print("No sqlite db found")

    study2=create_study(study_name=studyname,direction='maximize',storage=f'sqlite:///{storage.as_posix()}',load_if_exists=False) #TPESampler used as default, no pruning
    study2.optimize(lambda trial: objective(trial,xtrain),n_trials=25,n_jobs=-1,show_progress_bar=True)
    joblib.dump(study,thismodelpath.joinpath(f'study_{study.study_name}.pkl').as_posix())

    model = inst_pipeTFLDA(study.best_trial)
    fitpipe=model.fit(fulldatadf.description)
    joblib.dump(fitpipe,thismodelpath.joinpath(f'fittedpipelinedefault2.pkl').as_posix())
else:
    fitpipe=joblib.load(thismodelpath.joinpath(f'fittedpipelinedefault2.pkl').as_posix())
```

```
Directory already exists
[I 2024-09-08 15:48:56,204] A new study created in RDB with name: description_countlda_study
Best trial: 3. Best value: 0.637635:   4%|▍          | 1/25 [03:02<1:12:53, 182.21s/it]
[I 2024-09-08 15:51:58,378] Trial 3 finished with value: 0.6376348382855158 and parameters: {'stop_words': 'english', 'n_components': 6}. Best is trial 3 with value: 0.6376348382855158.
Best trial: 3. Best value: 0.637635:   8%|▊          | 2/25 [03:04<29:21, 76.57s/it]
[I 2024-09-08 15:52:00,931] Trial 11 finished with value: 0.6336377555448636 and parameters: {'stop_words': 'english', 'n_components': 4}. Best is trial 3 with value: 0.6376348382855158.
[I 2024-09-08 15:52:01,017] Trial 9 finished with value: 0.6300417038535931 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 20}. Best is trial 3 with value: 0.6376348382855158.
Best trial: 3. Best value: 0.637635:  16%|█▌         | 4/25 [03:16<11:15, 32.16s/it]
```

[I 2024-09-08 15:52:12,479] Trial 8 finished with value: 0.6325490289277648 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 10}. Best is trial 3 with value: 0.6376348382855158.

Best trial: 3. Best value: 0.637635: 16%|█ | 4/25 [03:44<11:15, 32.16s/it]

[I 2024-09-08 15:52:40,312] Trial 2 finished with value: 0.6276068090115572 and parameters: {'stop_words': 'english', 'n_components': 13}. Best is trial 3 with value: 0.6376348382855158.

Best trial: 3. Best value: 0.653039: 20%|██ | 5/25 [03:44<10:18, 30.93s/it]

[I 2024-09-08 15:52:40,823] Trial 14 finished with value: 0.6530389006334407 and parameters: {'stop_words': 'english', 'n_components': 3}. Best is trial 14 with value: 0.6530389006334407.

Best trial: 3. Best value: 0.653039: 20%|██ | 5/25 [03:45<10:18, 30.93s/it]

[I 2024-09-08 15:52:41,351] Trial 4 finished with value: 0.6301439060530812 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 9}. Best is trial 14 with value: 0.6530389006334407.

Best trial: 14. Best value: 0.653039: 28%|███ | 7/25 [03:48<04:34, 15.22s/it]

[I 2024-09-08 15:52:45,005] Trial 10 finished with value: 0.6325490289277648 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 10}. Best is trial 14 with value: 0.6530389006334407.

Best trial: 14. Best value: 0.653039: 32%|████ | 8/25 [03:49<03:18, 11.68s/it]

[I 2024-09-08 15:52:45,552] Trial 12 finished with value: 0.6331332192675154 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 13}. Best is trial 14 with value: 0.6530389006334407.

[I 2024-09-08 15:52:45,564] Trial 7 finished with value: 0.6286677394681384 and parameters: {'stop_words': 'english', 'n_components': 15}. Best is trial 14 with value: 0.6530389006334407.

[I 2024-09-08 15:52:45,665] Trial 1 finished with value: 0.6397787688139063 and parameters: {'stop_words': 'english', 'n_components': 5}. Best is trial 14 with value: 0.6530389006334407.

Best trial: 14. Best value: 0.653039: 48%|████▊ | 12/25 [03:49<00:47, 3.65s/it]

[I 2024-09-08 15:52:46,015] Trial 5 finished with value: 0.6317944601302492 and parameters: {'stop_words': 'english', 'n_components': 17}. Best is trial 14 with value: 0.6530389006334407.

[I 2024-09-08 15:52:46,026] Trial 13 finished with value: 0.6300417038535931 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 20}. Best is trial 14 with value: 0.6530389006334407.

Best trial: 14. Best value: 0.653039: 56%|█████ | 14/25 [03:50<00:26, 2.42s/it]

```
[I 2024-09-08 15:52:46,208] Trial 15 finished with value: 0.6341456332127604 and parameters: {'stop_words': 'english', 'n_components': 19}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:52:46,240] Trial 6 finished with value: 0.6321756997589514 and parameters: {'stop_words': 'english', 'n_components': 16}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:52:46,248] Trial 0 finished with value: 0.6300417038535931 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 20}. Best is trial 14 with value: 0.6530389006334407.
Best trial: 14. Best value: 0.653039:  76%|███████   | 19/25 [05:33<01:36, 16.07s/it]
[I 2024-09-08 15:54:29,209] Trial 17 finished with value: 0.6371556698621996 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 5}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:54:29,228] Trial 16 finished with value: 0.639662801021175 and parameters: {'stop_words': 'english', 'n_components': 10}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:54:29,243] Trial 19 finished with value: 0.6268309098521658 and parameters: {'stop_words': ['my', 'bill', 'him', 'whose', 'keep', 'hereafter', 'we', 'doesn', 'as', 'regarding', 'further', 'somehow', 'throughout', 'himself', 'somewhere', 'hers', 'during', 'co', 'whither', 'because', 'off', 'computer', 'once', 'no', 'out', 'amoungst', 'who', 'being', 'bottom', 'side', 'though', 'system', 'amount', 'if', 'detail', 'each', 'whenever', 'fire', 'few', 'put', 'neither', 'same', 'amongst', 'hereby', 'of', 'then', 'also', 'below', 'too', 'so', 'even', 'cant', 'show', 'thin', 'me', 'eg', 'here', 'beforehand', 'done', 'least', 'again', 'see', 'last', 'don', 'cannot', 'whole', 'latter', 'namely', 'everyone', 'your', 'everything', 'ie', 'became', 'about', 'often', 'via', 'full', 'behind', 'etc', 'please', 'four', 'own', 'thereby', 'give', 'toward', 'although', 'otherwise', 'nevertheless', 'most', 'when', 'has', 'latterly', 'nobody', 'you', 'become', 'sometime', 'with', 'yourself', 'part', 'itself', 'make', 'its', 'our', 'eleven', 'is', 'still', 'various', 'not', 'six', 'move', 'sometimes', 'either', 'go', 'they', 'anyone', 'fifteen', 'yours', 'afterwards', 'how', 'other', 'some', 'by', 'using', 'along', 'yourselves', 'there', 'whereby', 'his', 'thru', 'anything', 'wherein', 'towards', 'everywhere', 'ours', 'than', 'formerly', 'from', 'be', 'nine', 'one', 'that', 'over', 'seem', 'noone', 'in', 'empty', 'made', 'call', 'becoming', 'sixty', 'but', 'had', 'really', 'and', 'wherever', 'others', 'upon', 'hereupon', 'first', 'thereupon', 'does', 'only', 'alone', 'where', 'whether', 'mostly', 'at', 'can', 'after', 'whatever', 'hence', 'would', 'without', 'them', 'twelve', 'former', 'any', 'therein', 'sincere', 'was', 'hasnt', 'to', 'such', 'whence', 'rather', 'under', 'whereafter', 'third', 'now', 'un', 'which', 'forty', 'an', 'ten', 'nor', 'into', 'someone', 'whereas', 'thus', 'anyway', 'top', 'nothing', 'de', 'through', 'mine', 'moreover', 'before', 'herself', 'quite', 'almost', 'find', 'perhaps', 'beside', 'name', 'nowhere', 'except', 'three', 'twenty', 'why', 'may', 'herein', 'while', 'thereafter', 'several', 'he', 'much', 'what', 'down', 'am', 'yet', 'within', 'were', 'well', 'seems', 'are', 'the', 'together', 'anywhere', 'two', 'many', 'always', 'used', 'say', 'none', 'or', 'could', 'doing', 'next', 'she', 'else', 'ever', 'front', 'beyond', 'above', 'five', 'up', 'elsewhere', 'km', 'never', 'every', 'hundred', 'anyhow', 'whom', 'against', 'i', 'us', 'should', 'describe', 'across', 'cry', 'fifty', 'her', 'for', 'interest', 'enough', 'did', 'due', 'around', 'however', 'it', 'themselves', 'another', 'these', 'didn', 'whereupon', 'on', 're', 'mill', 'something', 'back', 'found', 'inc', 'this', 'serious', 'among', 'take', 'per', 'ourselves', 'unless', 'their', 'thick', 'will', 'couldnt', 'myself', 'meanwhile', 'con', 'been', 'onto', 'ltd', 'eight', 'becomes', 'seeming', 'less', 'must', 'a', 'whoever', 'since', 'both', 'very', 'thence', 'seemed', 'until', 'between', 'do', 'those', 'besides', 'more', 'already', 'therefore', 'indeed', 'all', 'kg', 'get', 'have', 'fill', 'might', 'just'], 'n_components': 6}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:54:29,247] Trial 18 finished with value: 0.6276068090115572 and parameters: {'stop_words': 'english', 'n_components': 13}. Best is trial 14 with value: 0.6530389006334407.
Best trial: 14. Best value: 0.653039: 100%|████████| 25/25 [05:51<00:00, 14.08s/it]
[I 2024-09-08 15:54:47,840] Trial 20 finished with value: 0.6376348382855158 and parameters: {'stop_words': 'english', 'n_components': 6}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:54:47,981] Trial 21 finished with value: 0.6530389006334407 and parameters: {'stop_words': 'english', 'n_components': 3}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:54:48,012] Trial 24 finished with value: 0.6530389006334407 and parameters: {'stop_words': 'english', 'n_components': 3}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:54:48,045] Trial 23 finished with value: 0.6530389006334407 and parameters: {'stop_words': 'english', 'n_components': 3}. Best is trial 14 with value: 0.6530389006334407.
[I 2024-09-08 15:54:48,075] Trial 22 finished with value: 0.6530389006334407 and parameters: {'stop_words': 'english', 'n_components': 3}. Best is trial 14 with value: 0.6530389006334407.
```

In [79]:
```
no_top_words=10
topics=fitpipe.named_steps['lda'].components_
for topic in topics:
    print(" ".join([fitpipe.named_steps['vectorizer'].get_feature_names_out()[i] for i in topic.argsort()[:-no_top_words - 1:-1]]))
```

```
s t life way new time summer long save heart
s world love life new old women year story family
s eragon new story world life t author harry dragon
```

lda with gridsearchcv and nmf as alternative

In [80]:
```
# optimizing all hyperparameters makes script run endlessly, does not converge
# no max features for vectorizers
# nmf has no scorer, no further tries, as to focus on lda
retrain_tfidflda=True
retrain_cvlda=True

cv=CountVectorizer(tokenizer=tokenize)
tfidf=TfidfVectorizer(tokenizer=tokenize)
lda=LatentDirichletAllocation(n_jobs=-1,random_state=randomstate)
nmf=NMF(random_state=randomstate)
paramgrida={'vectorizer__stop_words':['english',list(STOPWORDS)],
            # 'vectorizer__max_features':[None,100,50],
            'lda__n_components':[3,5],
            # 'lda__learning_method':['batch','online'],
            # 'lda__learning_decay':[0.5,0.9,0.7],
            # 'lda__learning_offset':[2,10],
            # 'lda__max_iter':[2,10],
            # 'lda__batch_size':[10,50],
            # 'lda__max_doc_update_iter':[1,5]
            }
pipelinea=Pipeline([('vectorizer', cv),('lda',lda)])
paramgridb={'vectorizer__norm':['l1','l2', None],
            'vectorizer__smooth_idf':[True,False],
            'vectorizer__sublinear_tf':[True,False],
            'vectorizer__stop_words':['english',list(STOPWORDS)],
            # 'vectorizer__max_features':[None,100,50],
            'lda__n_components':[3,5],
            # 'lda__learning_method':['batch','online'],
            # 'lda__learning_decay':[0.5,0.9,0.7],
            # 'lda__learning_offset':[2,10],
```

```
            # 'lda__max_iter':[2,10],
            # 'lda__batch_size':[10,50],
            # 'lda__max_doc_update_iter':[1,5]
            }
pipelineb=Pipeline([('vectorizer', tfidf),('lda',lda)])
#nmf has no scorer, can't be optimized unless specified
# paramgridc={'nmf__n_components':['auto',None,5,10,25,50,100],
#             'nmf__init':['random', 'nndsvd', 'nndsvda', 'nndsvdar'],
#             # 'nmf__solver':['cd','mu'],
#             # 'nmf__beta_loss':['frobenius', 'kullback-leibler', 'itakura-saito'],
#             # 'nmf__tol':[0.001,0.01,0.003,0.03],
#             # 'nmf__max_iter':[20,50,100,200],
#             # 'nmf__alpha_W':[0.0,0.5,1],
#             # 'nmf__alpha_H':[0.0,0.5,1],
#             # 'nmf__l1_ratio':[0.0,0.25,0.5],
#             # 'nmf__shuffle':[True,False],
#             'vectorizer__stop_words':['english',list(STOPWORDS)],
#             'vectorizer__max_features':[None,100,50,25,10]
#             }
# pipelinec=Pipeline([('vectorizer', cv),('nmf',nmf)])
# paramgridd={'nmf__n_components':['auto',None,5,10,25,50,100],
#             'nmf__init':['random', 'nndsvd', 'nndsvda', 'nndsvdar'],
#             # 'nmf__solver':['cd','mu'],
#             # 'nmf__beta_loss':['frobenius', 'kullback-leibler', 'itakura-saito'],
#             # 'nmf__tol':[0.001,0.01,0.003,0.03],
#             # 'nmf__max_iter':[20,50,100,200],
#             # 'nmf__alpha_W':[0.0,0.5,1],
#             # 'nmf__alpha_H':[0.0,0.5,1],
#             # 'nmf__l1_ratio':[0.0,0.25,0.5],
#             # 'nmf__shuffle':[True,False],
#             'vectorizer__norm':['l1','l2', None],
#             'vectorizer__smooth_idf':[True,False],
#             'vectorizer__sublinear_tf':[True,False],
#             'vectorizer__stop_words':['english',list(STOPWORDS)],
#             'vectorizer__max_features':[None,100,50,25,10]
#             }
# pipelined=Pipeline([('vectorizer', tfidf),('nmf',nmf)])
descriptionmodelspath=models_path.joinpath('description_models')

xtrain,xtest=train_test_split(fulldatadf['description'],test_size=0.1,random_state=randomstate,shuffle=True)
if retrain_cvlda:
    gridsearcha=GridSearchCV(pipelinea,paramgrida,n_jobs=-1,cv=5,return_train_score=True)
    gridamodel=gridsearcha.fit(xtrain)
    joblib.dump(gridamodel,descriptionmodelspath.joinpath('gridamodel.pkl').as_posix())
else:
    gridamodel=joblib.load(descriptionmodelspath.joinpath('gridamodel.pkl').as_posix())

if retrain_tfidflda:
    gridsearchb=GridSearchCV(pipelineb,paramgridb,n_jobs=-1,cv=5,return_train_score=True)
    gridbmodel=gridsearchb.fit(xtrain)
    joblib.dump(gridbmodel,descriptionmodelspath.joinpath('gridbmodel.pkl').as_posix())
else:
    gridbmodel=joblib.load(descriptionmodelspath.joinpath('gridbmodel.pkl').as_posix())

# gridsearchc=GridSearchCV(pipelinec,paramgridc,n_jobs=-1,cv=5,return_train_score=True)
# gridcmodel=gridsearchc.fit(xtrain)
# joblib.dump(gridcmodel,descriptionmodelspath.joinpath('gridcmodel.pkl').as_posix())

# gridsearchd=GridSearchCV(pipelined,paramgridd,n_jobs=-1,cv=5,return_train_score=True)
# griddmodel=gridsearchd.fit(xtrain)
# joblib.dump(griddmodel,descriptionmodelspath.joinpath('griddmodel.pkl').as_posix())
```

In [82]:
```
no_top_words=10
topics=gridamodel.best_estimator_.named_steps['lda'].components_
for topic in topics:
    print(" ".join([fitpipe.named_steps['vectorizer'].get_feature_names_out()[i] for i in topic.argsort()[:-no_top_words - 1:-1]]))
```

```
quinn trail jacobi marr july treat kiss theatres michacomes bosses
quinn jacobi gerard open trail marr security kiss silenced touched
quinn silenced dwarves jacobi marr incest injure treat trail security
```

In [83]:
```
no_top_words=10
topics=gridbmodel.best_estimator_.named_steps['lda'].components_
for topic in topics:
    print(" ".join([fitpipe.named_steps['vectorizer'].get_feature_names_out()[i] for i in topic.argsort()[:-no_top_words - 1:-1]]))
```

```
quinn afford trail opposing marr silenced loosed jacobi romance maintain
quinn security jacobi marr citadel hair granddaughter kiss stopping romance
quinn keeps grow afterlife jacobi silenced kiss spots trail fears
```

Insights and Findings

Topic Modeling (Lea)

- The trained topic models often created uniformly distributed topics across the documents, as the same tokens appeared in all topics. This leads to unusable features for prediction. Choosing any maximum value for tokens used by vectorizer results in this.
- Setting most hyperparameters to a value unequal their default seems to be an issue. Due to timecontraints for the creation of this assignment it was not possible to evaluate the cause of this and optimize accordingly. Most hyperparameters were left at default values as a result.
- Initially the coherence metric was supposed to be used for the evaluation of topics, but the combination of maximising coherence via optuna and crossvalidation via scikit's KFold ran into errors. Only later, after the rating prediction models were being trained, have I gotten used to working with optuna for hyperparameter optimization and fixed the code. Evaluating via scikit's cross_val_score requires a scorer (created via make_scorer) which only works for supervised problems, that have a target value.
- Learning to use optuna and combining it with correct crossvalidation and scoring has cost valuable time for this project.
- In the end, it was decided to continue with a pipeline optimized with GridSearchCV and the models default score.
- NMF Models were mentioned as an alternative to LDA in a couple of sources read. The models were imprelemted, as seen in the code, but as they lack a default scorer in scikit, fitting ran into an error. As I still struggeled with topic evaluation metrics and the make_scorer function, it was decided to not follow this path further and focus on the rating predictions.
- When early model fitting tries resulted in objectively bad models while using TFIDF vectors, the switch to CountVectorizer vectors was made. Research showed that some LDA implementations would only take CV-vectors. Models weren't improving and Scikit's own demos showed TFIDF being used. At a later date it was found, that a hyperparameter was set to float instead if int and didn't throw obvious errors. Time was lost.
- After all tries, the resulting topic models are not yet satisfactory. 3 Topics per model seem to be optimal for the data, but the topics are not destinct enough for my liking. This could be because the blurbs are not as distinct as they could be and keeping the books contents very vague. Further analysis would be required. The number of distinct books could also be increased. During exploration it was found, that the book metadata are directly lifted from goodreads.com and that the book_titles.csv table included in the dataset contains goodreads IDs which could be used to write a crawler to obtain more book metadata. This was not followed due to time constraints.
- Further the influence of text data preprocessing methods like lemmatization and stemming should also be evaluated.

Predicting user ratings

- based on context information about books
- based on information about users that have rated
- Insights and Findings

Predicting ratings based on topic modeling for books

In [96]:
```
datawreviewsfile=datadirpath.joinpath('processed/innerJoinData.csv')
datadf=pd.read_csv(datawreviewsfile,sep=';')

datadf=datadf.drop(['biography', 'children', 'comics', 'crime', 'fantasy', 'fiction',
       'graphic', 'historicalfiction', 'history', 'mystery', 'nonfiction',
       'paranormal', 'poetry', 'romance', 'thriller', 'youngadult'],axis=1)


datadf['description']=datadf.description.transform(remove_punctuation)
datadf['genre']=datadf.genre.apply(lambda x: x.replace("'",")) # remove '
datadf['genre']=datadf.genre.apply(lambda x: x.replace("-",")) # revome -
datadf['genre']=datadf.genre.apply(lambda x: x.replace(" ",")) # remove spaces
datadf['genre']=datadf.genre.apply(lambda x: x[1:-1].split(',')) # split into list

# encode genres
mlbmodelpath=models_path.joinpath('mlb_model')
mlbmodel=joblib.load(mlbmodelpath.joinpath('mlbmodel.pkl').as_posix())
newcols=mlbmodel.classes_
pred=mlbmodel.transform(datadf.genre)
preddf=pd.DataFrame(pred,columns=newcols)
datadf=datadf.join(preddf)

# encode description
descriptionmodelpath=models_path.joinpath('description_models')
# descmodel=joblib.load(descriptionmodelpath.joinpath('gridamodel.pkl').as_posix())
descmodel=joblib.load(models_path.joinpath('description_tfidflda_study/fitpipelinedefault.pkl').as_posix())
newcols=descmodel.named_steps['lda'].get_feature_names_out()
pred=descmodel.transform(datadf.description)
preddf=pd.DataFrame(pred,columns=newcols)
datadf=datadf.join(preddf)
```

In [97]:
```
# datadf.head()
datadf.columns
```

Out[97]:
```
Index(['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping',
       'Predicted Rating', 'Actual Rating', 'book_id_y', 'title_y',
       'image_url', 'url', 'num_pages', 'ratings_count', 'description',
       'genre', 'name', 'num_genres', 'biography', 'children', 'comics',
       'crime', 'fantasy', 'fiction', 'graphic', 'historicalfiction',
       'history', 'mystery', 'nonfiction', 'paranormal', 'poetry', 'romance',
       'thriller', 'youngadult', 'latentdirichletallocation0',
       'latentdirichletallocation1', 'latentdirichletallocation2'],
      dtype='object')
```

In [98]:
```
# drop columns not for prediction
# copied list:
# ['title_x','book_id_x','book_id_mapping','Predicted Rating','book_id_y', 'title_y', 'image_url', 'url','description', 'genre','num_genres']
# keep:'user_id_mapping' for user identification --> would be replaced by cluster representation, concern: as a numerical value implies false relationship between values
# target: 'Actual Rating'
dropcols=['title_x','book_id_x','book_id_mapping','Predicted Rating','book_id_y', 'title_y', 'image_url', 'url','description', 'genre','num_genres','name'] #drop author name too,
datadf=datadf.drop(dropcols,axis=1)
```

In [99]:
```
xtrain,xtest,ytrain,ytest=train_test_split(datadf.drop(['Actual Rating'],axis=1),datadf['Actual Rating'],test_size=0.1,random_state=randomstate,shuffle=True)
print(len(xtrain), len(ytrain))
print(len(xtest),len(ytest))
xtrain.dtypes
```

```
17841 17841
1983 1983
```

Out[99]:
```
user_id_mapping                int64
num_pages                      int64
ratings_count                  int64
biography                      int32
children                       int32
comics                         int32
crime                          int32
fantasy                        int32
fiction                        int32
graphic                        int32
historicalfiction              int32
history                        int32
mystery                        int32
nonfiction                     int32
paranormal                     int32
poetry                         int32
romance                        int32
thriller                       int32
youngadult                     int32
latentdirichletallocation0   float64
latentdirichletallocation1   float64
latentdirichletallocation2   float64
dtype: object
```

Regression

In [100…
```
predictionmodelpath=models_path.joinpath('prediction_models')
```

In [101…
```
modela=svm.SVR(max_iter=150)
paramsa={'kernel':['linear','poly','rbf','sigmoid'],
         'degree':[3,4],
         'gamma':['auto','scale'],
         'coef0':[0.0,0.5,1],
         'tol':[0.001,0.003,0.01],
         'C':[1.0,0.5,2.0],
         'epsilon':[0.1,0.5],
         'shrinking':[True,False]
         }
grida=GridSearchCV(modela,paramsa,n_jobs=-1,cv=5,return_train_score=True)

modelb=DecisionTreeRegressor(random_state=randomstate)
paramsb={'criterion':['squared_error','friedman_mse','absolute_error','poisson'],
         'splitter':['best','random'],
         'max_depth':[None,5,10],
         'min_samples_split':[2,25,100],
         'min_samples_leaf':[1,100,50],
         'max_features':[None,'log2',5],
         'max_leaf_nodes':[None,10,20],
         'ccp_alpha':[0.0,0.2,0.5]
         }
gridb=GridSearchCV(modelb,paramsb,n_jobs=-1,cv=5,return_train_score=True)
```

In [102…
```
retrain_svr=True
retrain_regtree=True
if retrain_svr:
```

```
    regsvmmodel=grida.fit(xtrain,ytrain)
    joblib.dump(regsvmmodel,predictionmodelpath.joinpath('regsvmmodel.pkl').as_posix())
else:
    regsvmmodel=joblib.load(predictionmodelpath.joinpath('regsvmmodel.pkl').as_posix())
if retrain_regtree:
    regdtmodel=gridb.fit(xtrain,ytrain)
    joblib.dump(regdtmodel,predictionmodelpath.joinpath('regdtmodel.pkl').as_posix())
else:
    regdtmodel=joblib.load(predictionmodelpath.joinpath('regdtmodel.pkl').as_posix())
```

Classification

In [103…

```
modelc=SVC(max_iter=150,random_state=randomstate)
paramsc={
    'C':[1.0,0.5,2.0],
    'kernel':['linear','poly','rbf','sigmoid'],
    'degree':[3,4],
    'gamma':['auto','scale'],
    'coef0':[0.0,0.5,1],
    'tol':[0.001,0.003,0.01],
    'shrinking':[True,False],
    'decision_function_shape':['ovo','ovr']
}
gridc=GridSearchCV(modelc,paramsc,n_jobs=-1,cv=5,return_train_score=True)

modeld=DecisionTreeClassifier(random_state=randomstate)
paramsd={
    'criterion':['gini','entropy','log_loss'],
    'splitter':['best','random'],
    'max_depth':[None,5,10],
    'min_samples_split':[2,25,100],
    'min_samples_leaf':[1,100,50],
    'max_features':[None,'log2',5],
    'max_leaf_nodes':[None,10,20],
    'ccp_alpha':[0.0,0.2,0.5]
}
gridd=GridSearchCV(modeld,paramsd,n_jobs=-1,cv=5,return_train_score=True)
```

In [104…

```
retrain_svm=True # runs vergy long, reload model as default, maybe use scaler
retrain_clastree=True
if retrain_svm:
    classvmmodel=gridc.fit(xtrain,ytrain)
    joblib.dump(classvmmodel,predictionmodelpath.joinpath('classsvmmodel.pkl').as_posix())
else:
    classvmmodel=joblib.load(predictionmodelpath.joinpath('classsvmmodel.pkl').as_posix())

if retrain_clastree:
    clasdtmodel=gridd.fit(xtrain,ytrain)
    joblib.dump(clasdtmodel,predictionmodelpath.joinpath('clasdtmodel.pkl').as_posix())
else:
    clasdtmodel=joblib.load(predictionmodelpath.joinpath('clasdtmodel.pkl').as_posix())
```

evaluate models

In [105…

```
predregsvm=regsvmmodel.predict(xtest)
predregdt=regdtmodel.predict(xtest)

msesvm=mean_squared_error(ytest,predregsvm)
msedt=mean_squared_error(ytest,predregdt)
r2svm=r2_score(ytest,predregsvm)
r2dt=r2_score(ytest,predregdt)
```

In [106…

```
print(f'MSE SVM: {msesvm}, MSE DT: {msedt}')
print(f'R2 SVM: {r2svm}, R2 DT: {r2dt}')
```

```
MSE SVM: 1.3435918308909056, MSE DT: 1.0309330023640264
R2 SVM: -0.20169662014646916, R2 DT: 0.07794266379482506
```

In [107…

```
predclassvm=classvmmodel.predict(xtest)
predclasdt=clasdtmodel.predict(xtest)

accsvm=accuracy_score(ytest,predclassvm)
accdt=accuracy_score(ytest,predclasdt)
msesvm2=mean_squared_error(ytest,predclassvm)
msedt2=mean_squared_error(ytest,predclasdt)
```

In [108…

```
print(f'Accuracy SVM: {accsvm}, Accuracy DT: {accdt}') # the accuracy is not good, models might only predict a rating of 4 or 5 (unbalanced classes)
print(f'MSE SVM: {msesvm2}, MSE DT: {msedt2}')
```

```
Accuracy SVM: 0.3398890569843671, Accuracy DT: 0.37065052950075644
MSE SVM: 2.017650025214322, MSE DT: 1.312657589510842
```

Regression and Classification NN

In [109…

```
retrain_studyregression=True
xtrain,xtest,ytrain,ytest=train_test_split(datadf.drop(['Actual Rating'],axis=1),datadf['Actual Rating'],test_size=0.1,random_state=randomstate,shuffle=True)
xtrain,xtest,ytrain,ytest=xtrain.reset_index(drop=True),xtest.reset_index(drop=True),ytrain.reset_index(drop=True),ytest.reset_index(drop=True) #kf.split() needs reset index; if
studyname='prediction_regression'
thismodelpath=models_path.joinpath(f'{studyname}')
try:
    thismodelpath.mkdir(exist_ok=False)
except FileExistsError:
    print('Directory already exists')

if retrain_studyregression:
    def init_regmodel(trial):
        # model definition
        nlayers=trial.suggest_int('n_layers',1,10)
        model=tf.keras.Sequential()
        activation=trial.suggest_categorical('activation',['relu','tanh','sigmoid'])
        inputshape=xtrain.shape[1]
        model.add(tf.keras.layers.Dense(16, input_shape=(inputshape,),activation=activation))
        for i in range(nlayers):
            numhidden=trial.suggest_int(f'n_units_l{i}',32,128,log=True)
            model.add(tf.keras.layers.Dense(units=numhidden,activation=activation,name=f'layer{i}'))
        model.add(tf.keras.layers.Dense(units=1,activation='linear'))
        model.compile(optimizer='adam', loss='mean_absolute_error', metrics=['mean_squared_error'])
        return model

    def reg_objective(trial:Trial,xtrain:pd.DataFrame,ytrain:pd.Series)->float:
        kf = KFold(n_splits=5, shuffle=True, random_state=randomstate)
        model=init_regmodel(trial)
        # training and evaluation
        msetest=[]
        for train,test in kf.split(xtrain):
            trdata=tf.convert_to_tensor(xtrain.loc[train])
```

```
            trlabel=tf.convert_to_tensor(ytrain.loc[train])
            evdata=tf.convert_to_tensor(xtrain.loc[test])
            evlabel=tf.convert_to_tensor(ytrain.loc[test])

            model.fit(trdata,trlabel,validation_split=0.1,epochs=10, batch_size=128,verbose=0) #more verbosity seems to crash in execution

            loss,mse=model.evaluate(evdata, evlabel)
            # predtrain=tmodel.predict(evdata)
            # mse=tf.keras.losses.MSE(evlabel, predtrain)
            msetest.append(mse)

        return np.mean(msetest)

    storage=thismodelpath.joinpath(f"{studyname}.db")
    if storage.exists():
        storage.unlink()
    else:
        print("No sqlite db found")

    study=create_study(study_name=studyname,direction='minimize',storage=f'sqlite:///{storage.as_posix()}',load_if_exists=False)
    study.optimize(lambda trial: reg_objective(trial,xtrain,ytrain),n_trials=20,n_jobs=-1,show_progress_bar=True)

    joblib.dump(study,thismodelpath.joinpath(f'study_{study.study_name}.pkl').as_posix())

    regmodel = init_regmodel(study.best_trial)
    regmodel.fit(tf.convert_to_tensor(xtrain),tf.convert_to_tensor(ytrain),validation_split=0.1,epochs=10, batch_size=128,verbose=0)
    regmodel.save(thismodelpath.joinpath(f'fmodel_reg.h5').as_posix())

else:
    regmodel = tf.keras.models.load_model(thismodelpath.joinpath(f'fmodel_reg.h5').as_posix())
```

```
Directory already exists
[I 2024-09-08 17:36:41,168] A new study created in RDB with name: prediction_regression
  0%|          | 0/20 [00:00<?, ?it/s]
112/112 [==============================] - 2s 13ms/step - loss: 0.7746 - mean_squared_error: 1.0652
112/112 [==============================] - 2s 18ms/step - loss: 0.7702 - mean_squared_error: 1.0769
112/112 [==============================] - 2s 17ms/step - loss: 0.7659 - mean_squared_error: 1.0739
112/112 [==============================] - 2s 18ms/step - loss: 0.7978 - mean_squared_error: 1.0713
112/112 [==============================] - 2s 17ms/step - loss: 0.7716 - mean_squared_error: 1.0781
112/112 [==============================] - 2s 16ms/step - loss: 0.7672 - mean_squared_error: 1.0698
112/112 [==============================] - 2s 18ms/step - loss: 0.7672 - mean_squared_error: 1.0691
112/112 [==============================] - 2s 21ms/step - loss: 0.7653 - mean_squared_error: 1.0736
112/112 [==============================] - 2s 21ms/step - loss: 0.7863 - mean_squared_error: 1.0918
112/112 [==============================] - 2s 19ms/step - loss: 0.7692 - mean_squared_error: 1.0762
112/112 [==============================] - 2s 17ms/step - loss: 275.1039 - mean_squared_error: 115593.8516
112/112 [==============================] - 2s 18ms/step - loss: 14.1554 - mean_squared_error: 260.4863
112/112 [==============================] - 2s 18ms/step - loss: 0.8452 - mean_squared_error: 1.0840
112/112 [==============================] - 2s 18ms/step - loss: 0.7659 - mean_squared_error: 1.0704
112/112 [==============================] - 2s 17ms/step - loss: 0.7661 - mean_squared_error: 1.0740
112/112 [==============================] - 2s 16ms/step - loss: 0.7696 - mean_squared_error: 1.0766
112/112 [==============================] - 2s 13ms/step - loss: 0.7744 - mean_squared_error: 1.0599
112/112 [==============================] - 2s 18ms/step - loss: 0.7562 - mean_squared_error: 1.0704
112/112 [==============================] - 2s 19ms/step - loss: 0.7615 - mean_squared_error: 1.0665
112/112 [==============================] - 2s 19ms/step - loss: 0.7570 - mean_squared_error: 1.0708
112/112 [==============================] - 2s 18ms/step - loss: 0.7599 - mean_squared_error: 1.0729
112/112 [==============================] - 2s 18ms/step - loss: 0.7564 - mean_squared_error: 1.0696
112/112 [==============================] - 3s 22ms/step - loss: 0.7610 - mean_squared_error: 1.0738
112/112 [==============================] - 2s 22ms/step - loss: 0.7568 - mean_squared_error: 1.0708
112/112 [==============================] - 2s 20ms/step - loss: 0.7671 - mean_squared_error: 1.0639
112/112 [==============================] - 2s 19ms/step - loss: 174.2915 - mean_squared_error: 43454.4453
112/112 [==============================] - 2s 18ms/step - loss: 4.5932 - mean_squared_error: 44.3054
112/112 [==============================] - 2s 21ms/step - loss: 0.7655 - mean_squared_error: 1.0645
112/112 [==============================] - 2s 18ms/step - loss: 0.7637 - mean_squared_error: 1.0759
112/112 [==============================] - 2s 20ms/step - loss: 0.8502 - mean_squared_error: 1.1291
112/112 [==============================] - 2s 19ms/step - loss: 0.7616 - mean_squared_error: 1.0665
112/112 [==============================] - 2s 16ms/step - loss: 0.7848 - mean_squared_error: 1.0574
112/112 [==============================] - 2s 14ms/step - loss: 0.7920 - mean_squared_error: 1.1024
112/112 [==============================] - 2s 19ms/step - loss: 0.7898 - mean_squared_error: 1.1041
112/112 [==============================] - 2s 18ms/step - loss: 0.7898 - mean_squared_error: 1.1037
112/112 [==============================] - 2s 17ms/step - loss: 0.7946 - mean_squared_error: 1.1113
112/112 [==============================] - 2s 18ms/step - loss: 0.8039 - mean_squared_error: 1.0969
112/112 [==============================] - 2s 21ms/step - loss: 0.7881 - mean_squared_error: 1.1051
112/112 [==============================] - 2s 20ms/step - loss: 0.7927 - mean_squared_error: 1.1022
112/112 [==============================] - 3s 22ms/step - loss: 0.7876 - mean_squared_error: 1.1059
112/112 [==============================] - 2s 19ms/step - loss: 14.7906 - mean_squared_error: 381.4174
112/112 [==============================] - 2s 19ms/step - loss: 1.3759 - mean_squared_error: 2.9899
112/112 [==============================] - 2s 21ms/step - loss: 0.7914 - mean_squared_error: 1.1087
112/112 [==============================] - 2s 18ms/step - loss: 0.7904 - mean_squared_error: 1.1036
112/112 [==============================] - 2s 21ms/step - loss: 0.7896 - mean_squared_error: 1.1074
112/112 [==============================] - 2s 17ms/step - loss: 0.7939 - mean_squared_error: 1.1108
112/112 [==============================] - 2s 18ms/step - loss: 0.8040 - mean_squared_error: 1.0979
112/112 [==============================] - 2s 16ms/step - loss: 0.7969 - mean_squared_error: 1.1133
112/112 [==============================] - 2s 15ms/step - loss: 0.8042 - mean_squared_error: 1.1525
112/112 [==============================] - 2s 17ms/step - loss: 0.8004 - mean_squared_error: 1.1492
112/112 [==============================] - 2s 17ms/step - loss: 0.8072 - mean_squared_error: 1.1554
112/112 [==============================] - 2s 19ms/step - loss: 0.8157 - mean_squared_error: 1.1350
112/112 [==============================] - 2s 18ms/step - loss: 0.8211 - mean_squared_error: 1.1734
112/112 [==============================] - 3s 25ms/step - loss: 0.7982 - mean_squared_error: 1.1473
112/112 [==============================] - 3s 25ms/step - loss: 0.7983 - mean_squared_error: 1.1471
112/112 [==============================] - 2s 18ms/step - loss: 0.8033 - mean_squared_error: 1.1429
112/112 [==============================] - 2s 16ms/step - loss: 1.0516 - mean_squared_error: 1.7671
112/112 [==============================] - 2s 19ms/step - loss: 0.8023 - mean_squared_error: 1.1436
112/112 [==============================] - 2s 17ms/step - loss: 0.7988 - mean_squared_error: 1.1464
112/112 [==============================] - 2s 17ms/step - loss: 67.1915 - mean_squared_error: 7193.7612
112/112 [==============================] - 2s 19ms/step - loss: 0.7990 - mean_squared_error: 1.1463
112/112 [==============================] - 2s 17ms/step - loss: 0.8020 - mean_squared_error: 1.1506
112/112 [==============================] - 2s 19ms/step - loss: 0.7982 - mean_squared_error: 1.1456
112/112 [==============================] - 2s 18ms/step - loss: 0.8059 - mean_squared_error: 1.1411
112/112 [==============================] - 2s 16ms/step - loss: 0.7973 - mean_squared_error: 1.1117
Best trial: 11. Best value: 1.09835:   5%|▌         | 1/20 [01:40<31:40, 100.03s/it]
[I 2024-09-08 17:38:20,996] Trial 11 finished with value: 1.0983523607254029 and parameters: {'n_layers': 1, 'activation': 'tanh', 'n_units_l0': 72}. Best is trial 11 with value:
1.0983523607254029.
112/112 [==============================] - 2s 18ms/step - loss: 0.7870 - mean_squared_error: 1.1203
 37/112 [========>.....................] - ETA: 1s - loss: 0.8038 - mean_squared_error: 1.1439
Best trial: 11. Best value: 1.09835:   5%|▌         | 1/20 [01:44<31:40, 100.03s/it]
[I 2024-09-08 17:38:25,275] Trial 9 finished with value: 1.1035995483398438 and parameters: {'n_layers': 2, 'activation': 'sigmoid', 'n_units_l0': 43, 'n_units_l1': 39}. Best is
trial 11 with value: 1.0983523607254029.
  3/112 [.............................] - ETA: 3s - loss: 0.8171 - mean_squared_error: 1.0661
Best trial: 11. Best value: 1.09835:   5%|▌         | 1/20 [01:44<31:40, 100.03s/it]
 49/112 [============>.................] - ETA: 1s - loss: 0.7888 - mean_squared_error: 1.1089
Best trial: 11. Best value: 1.09835:  10%|▉         | 2/20 [01:44<13:08, 43.78s/it]
112/112 [==============================] - 2s 21ms/step - loss: 0.7912 - mean_squared_error: 1.1236
112/112 [==============================] - 2s 19ms/step - loss: 0.8021 - mean_squared_error: 1.1340


 22/112 [====>.........................] - ETA: 1s - loss: 0.7976 - mean_squared_error: 1.0956
[I 2024-09-08 17:38:27,630] Trial 6 finished with value: 1.1082559823989868 and parameters: {'n_layers': 3, 'activation': 'sigmoid', 'n_units_l0': 71, 'n_units_l1': 60, 'n_units_
l2': 85}. Best is trial 11 with value: 1.0983523607254029.
Best trial: 11. Best value: 1.09835:  10%|▉         | 2/20 [01:46<13:08, 43.78s/it]
```

```
 96/112 [========================>.....] - ETA: 0s - loss: 0.7969 - mean_squared_error: 1.1065

 99/112 [========================>....] - ETA: 0s - loss: 0.7970 - mean_squared_error: 1.1110
Best trial: 11. Best value: 1.09835:  15%|█         | 3/20 [01:46<07:02, 24.86s/it]
[I 2024-09-08 17:38:27,937] Trial 1 finished with value: 1.1032121181488037 and parameters: {'n_layers': 3, 'activation': 'sigmoid', 'n_units_l0': 91, 'n_units_l1': 96, 'n_units_
l2': 108}. Best is trial 11 with value: 1.0983523607254029.
Best trial: 11. Best value: 1.09835:  15%|█         | 3/20 [01:46<07:02, 24.86s/it]
110/112 [============================>.] - ETA: 0s - loss: 0.7947 - mean_squared_error: 1.1105
Best trial: 11. Best value: 1.09835:  20%|██        | 4/20 [01:47<04:02, 15.16s/it]
112/112 [==============================] - 2s 19ms/step - loss: 0.7957 - mean_squared_error: 1.1136
112/112 [==============================] - 2s 20ms/step - loss: 0.7869 - mean_squared_error: 1.1202
 98/112 [=========================>....] - ETA: 0s - loss: 0.7875 - mean_squared_error: 1.1153

101/112 [==========================>...] - ETA: 0s - loss: 0.7886 - mean_squared_error: 1.1221
Best trial: 11. Best value: 1.09835:  20%|██        | 4/20 [01:49<04:02, 15.16s/it]
[I 2024-09-08 17:38:30,694] Trial 7 finished with value: 1.1052340030670167 and parameters: {'n_layers': 1, 'activation': 'tanh', 'n_units_l0': 118}. Best is trial 11 with value:
1.0983523607254029.
103/112 [==========================>...] - ETA: 0s - loss: 0.7869 - mean_squared_error: 1.1182

 72/112 [==================>...........] - ETA: 0s - loss: 0.7828 - mean_squared_error: 1.1100[I 2024-09-08 17:38:30,731] Trial 10 finished with value: 1.1024102449417115 and par
ameters: {'n_layers': 3, 'activation': 'tanh', 'n_units_l0': 102, 'n_units_l1': 106, 'n_units_l2': 34}. Best is trial 11 with value: 1.0983523607254029.
Best trial: 11. Best value: 1.09835:  20%|██        | 4/20 [01:49<04:02, 15.16s/it]
112/112 [==============================] - 2s 22ms/step - loss: 0.7867 - mean_squared_error: 1.1201
Best trial: 11. Best value: 1.09835:  25%|██        | 5/20 [01:49<02:41, 10.74s/it]
 87/112 [======================>.......] - ETA: 0s - loss: 0.7872 - mean_squared_error: 1.1126
Best trial: 11. Best value: 1.09835:  25%|██        | 5/20 [01:50<02:41, 10.74s/it]
 51/112 [============>.................] - ETA: 1s - loss: 1.0041 - mean_squared_error: 1.7279
Best trial: 11. Best value: 1.09835:  30%|███       | 6/20 [01:50<01:39,  7.12s/it]
 99/112 [=========================>....] - ETA: 0s - loss: 0.7878 - mean_squared_error: 1.1173

 62/112 [===============>..............] - ETA: 1s - loss: 0.9890 - mean_squared_error: 1.6756
[I 2024-09-08 17:38:31,590] Trial 3 finished with value: 1.103497052192688 and parameters: {'n_layers': 1, 'activation': 'sigmoid', 'n_units_l0': 41}. Best is trial 11 with valu
e: 1.0983523607254029.
  1/112 [..............................] - ETA: 7s - loss: 8.8016 - mean_squared_error: 155.2807
Best trial: 11. Best value: 1.09835:  30%|███       | 6/20 [01:50<01:39,  7.12s/it]
 11/112 [=>............................] - ETA: 2s - loss: 0.8078 - mean_squared_error: 1.0623
Best trial: 11. Best value: 1.09835:  30%|███       | 6/20 [01:50<01:39,  7.12s/it]
112/112 [==============================] - 3s 23ms/step - loss: 0.7864 - mean_squared_error: 1.1198
  9/112 [=>............................] - ETA: 2s - loss: 0.8199 - mean_squared_error: 1.0622
Best trial: 11. Best value: 1.09835:  35%|███▌      | 7/20 [01:50<01:05,  5.04s/it]
 32/112 [=======>......................] - ETA: 1s - loss: 0.7995 - mean_squared_error: 1.1243

 95/112 [========================>.....] - ETA: 0s - loss: 1.0112 - mean_squared_error: 1.74362
Best trial: 11. Best value: 1.09835:  35%|███▌      | 7/20 [01:51<01:05,  5.04s/it]
[I 2024-09-08 17:38:32,390] Trial 2 finished with value: 1.1015496969223022 and parameters: {'n_layers': 6, 'activation': 'tanh', 'n_units_l0': 85, 'n_units_l1': 102, 'n_units_l
2': 74, 'n_units_l3': 73, 'n_units_l4': 70, 'n_units_l5': 54}. Best is trial 11 with value: 1.0983523607254029.
 35/112 [========>.....................] - ETA: 1s - loss: 0.8041 - mean_squared_error: 1.1497
Best trial: 11. Best value: 1.09835:  40%|████      | 8/20 [01:51<00:43,  3.65s/it]
112/112 [==============================] - 3s 24ms/step - loss: 1.0103 - mean_squared_error: 1.7537
 70/112 [=================>............] - ETA: 0s - loss: 0.7857 - mean_squared_error: 1.10050

 62/112 [===============>..............] - ETA: 1s - loss: 10.1389 - mean_squared_error: 217.2094
[I 2024-09-08 17:38:33,141] Trial 0 finished with value: 62.260498070716866 and parameters: {'n_layers': 4, 'activation': 'relu', 'n_units_l0': 72, 'n_units_l1': 48, 'n_units_l2':
36, 'n_units_l3': 48}. Best is trial 11 with value: 1.0983523607254029.
Best trial: 11. Best value: 1.09835:  40%|████      | 8/20 [01:52<00:43,  3.65s/it]
 72/112 [==================>...........] - ETA: 0s - loss: 10.0932 - mean_squared_error: 214.0657
Best trial: 11. Best value: 1.09835:  40%|████      | 8/20 [01:52<00:43,  3.65s/it]
 83/112 [=====================>........] - ETA: 0s - loss: 0.7942 - mean_squared_error: 1.1172
Best trial: 11. Best value: 1.09835:  45%|████▌     | 9/20 [01:52<00:30,  2.76s/it]
112/112 [==============================] - 2s 20ms/step - loss: 0.7905 - mean_squared_error: 1.1164
112/112 [==============================] - 3s 24ms/step - loss: 10.1495 - mean_squared_error: 216.0366
112/112 [==============================] - 3s 23ms/step - loss: 0.7867 - mean_squared_error: 1.1194

  7/112 [>.............................] - ETA: 3s - loss: 0.8307 - mean_squared_error: 1.0717
Best trial: 11. Best value: 1.09835:  45%|████▌     | 9/20 [01:53<00:30,  2.76s/it]
[I 2024-09-08 17:38:34,431] Trial 4 finished with value: 1.102556324005127 and parameters: {'n_layers': 6, 'activation': 'sigmoid', 'n_units_l0': 46, 'n_units_l1': 100, 'n_units_
l2': 33, 'n_units_l3': 98, 'n_units_l4': 54, 'n_units_l5': 55}. Best is trial 11 with value: 1.0983523607254029.
 21/112 [====>.........................] - ETA: 2s - loss: 0.8114 - mean_squared_error: 1.1238
Best trial: 11. Best value: 1.09835:  45%|████▌     | 9/20 [01:53<00:30,  2.76s/it]
[I 2024-09-08 17:38:34,741] Trial 15 finished with value: 33367.90241394043 and parameters: {'n_layers': 2, 'activation': 'relu', 'n_units_l0': 59, 'n_units_l1': 122}. Best is tr
ial 11 with value: 1.0983523607254029.
 24/112 [=====>........................] - ETA: 2s - loss: 0.8064 - mean_squared_error: 1.1345

 26/112 [=====>........................] - ETA: 2s - loss: 0.7961 - mean_squared_error: 1.1181
Best trial: 11. Best value: 1.09835:  50%|█████     | 10/20 [01:53<00:23,  2.33s/it]
[I 2024-09-08 17:38:34,763] Trial 12 finished with value: 1.1054610013961792 and parameters: {'n_layers': 4, 'activation': 'tanh', 'n_units_l0': 90, 'n_units_l1': 60, 'n_units_l
2': 128, 'n_units_l3': 123}. Best is trial 11 with value: 1.0983523607254029.
 27/112 [======>.......................] - ETA: 2s - loss: 0.7875 - mean_squared_error: 1.1114
Best trial: 11. Best value: 1.09835:  55%|█████▌    | 11/20 [01:53<00:15,  1.68s/it]
 30/112 [=======>......................] - ETA: 2s - loss: 0.7921 - mean_squared_error: 1.1190
Best trial: 11. Best value: 1.09835:  55%|█████▌    | 11/20 [01:53<00:15,  1.68s/it]
112/112 [==============================] - 2s 19ms/step - loss: 0.7866 - mean_squared_error: 1.1200
105/112 [===========================>..] - ETA: 0s - loss: 0.7910 - mean_squared_error: 1.1193

109/112 [============================>.] - ETA: 0s - loss: 0.7893 - mean_squared_error: 1.1158
Best trial: 11. Best value: 1.09835:  60%|██████    | 12/20 [01:55<00:13,  1.68s/it]
[I 2024-09-08 17:38:36,591] Trial 8 finished with value: 1.1028719663619995 and parameters: {'n_layers': 5, 'activation': 'tanh', 'n_units_l0': 40, 'n_units_l1': 39, 'n_units_l
2': 123, 'n_units_l3': 93, 'n_units_l4': 57}. Best is trial 11 with value: 1.0983523607254029.
112/112 [==============================] - 2s 14ms/step - loss: 0.7893 - mean_squared_error: 1.1173
Best trial: 11. Best value: 1.09835:  70%|███████   | 14/20 [01:55<00:06,  1.04s/it]
[I 2024-09-08 17:38:36,914] Trial 14 finished with value: 1.1038359880447388 and parameters: {'n_layers': 8, 'activation': 'sigmoid', 'n_units_l0': 41, 'n_units_l1': 56, 'n_units
_l2': 53, 'n_units_l3': 34, 'n_units_l4': 34, 'n_units_l5': 38, 'n_units_l6': 51, 'n_units_l7': 37}. Best is trial 11 with value: 1.0983523607254029.
112/112 [==============================] - 1s 6ms/step - loss: 0.7871 - mean_squared_error: 1.1186
112/112 [==============================] - 1s 6ms/step - loss: 0.8058 - mean_squared_error: 1.1075

[I 2024-09-08 17:38:38,635] Trial 5 finished with value: 1.1150514841079713 and parameters: {'n_layers': 10, 'activation': 'relu', 'n_units_l0': 115, 'n_units_l1': 35, 'n_units_l
2': 43, 'n_units_l3': 69, 'n_units_l4': 77, 'n_units_l5': 41, 'n_units_l6': 96, 'n_units_l7': 89, 'n_units_l8': 120, 'n_units_l9': 73}. Best is trial 11 with value: 1.09835236072
54029.
Best trial: 11. Best value: 1.09835:  75%|███████▌  | 15/20 [01:57<00:06,  1.21s/it]
```

```
[I 2024-09-08 17:38:38,671] Trial 13 finished with value: 1.0992093324661254 and parameters: {'n_layers': 10, 'activation': 'sigmoid', 'n_units_l0': 46, 'n_units_l1': 99, 'n_units_l2': 61, 'n_units_l3': 40, 'n_units_l4': 68, 'n_units_l5': 116, 'n_units_l6': 112, 'n_units_l7': 77, 'n_units_l8': 108, 'n_units_l9': 128}. Best is trial 11 with value: 1.0983523607254029.
112/112 [==============================] - 0s 2ms/step - loss: 35.7304 - mean_squared_error: 1824.9718
112/112 [==============================] - 0s 3ms/step - loss: 0.7682 - mean_squared_error: 1.0754
112/112 [==============================] - 0s 3ms/step - loss: 1.9938 - mean_squared_error: 6.4383
112/112 [==============================] - 0s 3ms/step - loss: 1.5240 - mean_squared_error: 3.5206
112/112 [==============================] - 0s 2ms/step - loss: 14.6681 - mean_squared_error: 315.4272
112/112 [==============================] - 0s 2ms/step - loss: 0.7559 - mean_squared_error: 1.0701
112/112 [==============================] - 0s 2ms/step - loss: 1.1291 - mean_squared_error: 2.0381
112/112 [==============================] - 0s 3ms/step - loss: 17.4661 - mean_squared_error: 368.6435
112/112 [==============================] - 0s 3ms/step - loss: 1.7182 - mean_squared_error: 5.0679
112/112 [==============================] - 0s 2ms/step - loss: 0.7900 - mean_squared_error: 1.1075
112/112 [==============================] - 0s 3ms/step - loss: 7.0618 - mean_squared_error: 86.7387
112/112 [==============================] - 0s 3ms/step - loss: 1.5102 - mean_squared_error: 3.3377
112/112 [==============================] - 0s 2ms/step - loss: 0.9509 - mean_squared_error: 1.2514
112/112 [==============================] - 0s 2ms/step - loss: 0.8022 - mean_squared_error: 1.1507
112/112 [==============================] - 0s 2ms/step - loss: 8.9805 - mean_squared_error: 140.9207
Best trial: 11. Best value: 1.09835:  80%|████████  | 16/20 [02:13<00:04,  1.21s/it]
[I 2024-09-08 17:38:54,372] Trial 16 finished with value: 547.3403930664062 and parameters: {'n_layers': 2, 'activation': 'relu', 'n_units_l0': 66, 'n_units_l1': 82}. Best is trial 11 with value: 1.0983523607254029.
Best trial: 11. Best value: 1.09835:  85%|████████  | 17/20 [02:13<00:12,  4.06s/it]
112/112 [==============================] - 0s 2ms/step - loss: 0.9687 - mean_squared_error: 1.2961
112/112 [==============================] - 0s 2ms/step - loss: 1.0653 - mean_squared_error: 1.6794
112/112 [==============================] - 0s 3ms/step - loss: 0.7880 - mean_squared_error: 1.1182
Best trial: 11. Best value: 1.09835:  90%|████████  | 18/20 [02:16<00:07,  3.86s/it]
[I 2024-09-08 17:38:57,606] Trial 19 finished with value: 1.1043873310089112 and parameters: {'n_layers': 3, 'activation': 'sigmoid', 'n_units_l0': 74, 'n_units_l1': 56, 'n_units_l2': 55}. Best is trial 11 with value: 1.0983523607254029.
112/112 [==============================] - 0s 2ms/step - loss: 0.8575 - mean_squared_error: 1.2336
Best trial: 11. Best value: 1.09835:  90%|████████  | 18/20 [02:20<00:07,  3.86s/it]
[I 2024-09-08 17:39:01,734] Trial 17 finished with value: 2.868753504753113 and parameters: {'n_layers': 6, 'activation': 'relu', 'n_units_l0': 127, 'n_units_l1': 67, 'n_units_l2': 107, 'n_units_l3': 66, 'n_units_l4': 53, 'n_units_l5': 35}. Best is trial 11 with value: 1.0983523607254029.
Best trial: 11. Best value: 1.09835:  95%|████████  | 19/20 [02:20<00:03,  3.93s/it]
112/112 [==============================] - 0s 2ms/step - loss: 0.8243 - mean_squared_error: 1.1256
Best trial: 11. Best value: 1.09835: 100%|████████| 20/20 [02:22<00:00,  7.12s/it]
[I 2024-09-08 17:39:03,653] Trial 18 finished with value: 2.528973126411438 and parameters: {'n_layers': 7, 'activation': 'relu', 'n_units_l0': 86, 'n_units_l1': 83, 'n_units_l2': 111, 'n_units_l3': 53, 'n_units_l4': 84, 'n_units_l5': 34, 'n_units_l6': 32}. Best is trial 11 with value: 1.0983523607254029.
```

In [110…

```python
retrain_studyclassification=True
xtrain,xtest,ytrain,ytest=train_test_split(datadf.drop(['Actual Rating'],axis=1),datadf['Actual Rating'],test_size=0.1,random_state=randomstate,shuffle=True)
xtrain,xtest,ytrain,ytest=xtrain.reset_index(drop=True),xtest.reset_index(drop=True),ytrain.reset_index(drop=True),ytest.reset_index(drop=True)

studyname='prediction_classification'
thismodelpath=models_path.joinpath(f'{studyname}')
try:
    thismodelpath.mkdir(exist_ok=False)
except FileExistsError:
    print('Directory already exists')
if retrain_studyclassification:
    def init_clasmodel(trial):
        # model definition
        nlayers=trial.suggest_int('n_layers',1,10)
        model=tf.keras.Sequential()
        activation=trial.suggest_categorical('activation',['relu','tanh','sigmoid'])
        inputshape=xtrain.shape[1]
        model.add(tf.keras.layers.Dense(16, input_shape=(inputshape,),activation=activation))
        for i in range(nlayers):
            numhidden=trial.suggest_int(f'n_units_l{i}',32,128,log=True)
            model.add(tf.keras.layers.Dense(units=numhidden,activation=activation,name=f'layer{i}'))
        model.add(tf.keras.layers.Dense(units=5,activation='softmax'))
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        return model

    def clas_objective(trial:Trial,xtrain:pd.DataFrame,ytrain:pd.Series)->float:
        kf = KFold(n_splits=5, shuffle=True, random_state=randomstate)
        ytrain=datadf['Actual Rating'].apply(lambda x: x-1)
        model=init_clasmodel(trial)
        # training and evaluation
        acctest=[]
        for train,test in kf.split(xtrain):
            trdata=tf.convert_to_tensor(xtrain.loc[train])
            trlabel=tf.convert_to_tensor(tf.keras.utils.to_categorical(ytrain.loc[train],num_classes=5))
            evdata=tf.convert_to_tensor(xtrain.loc[test])
            evlabel=tf.convert_to_tensor(tf.keras.utils.to_categorical(ytrain.loc[test],num_classes=5))

            model.fit(trdata,trlabel,validation_split=0.1,epochs=10, batch_size=128,verbose=0) #more verbosity seems to crash in execution

            loss,acc=model.evaluate(evdata, evlabel)
            # predtrain=tmodel.predict(evdata)
            # mse=tf.keras.losses.MSE(evlabel, predtrain)
            acctest.append(acc)

        return np.mean(acctest)

    storage=thismodelpath.joinpath(f"{studyname}.db")
    if storage.exists():
        storage.unlink()
    else:
        print("No sqlite db found")

    study=create_study(study_name=studyname,direction='maximize',storage=f'sqlite:///{storage.as_posix()}',load_if_exists=False)
    study.optimize(lambda trial: clas_objective(trial,xtrain,ytrain),n_trials=20,n_jobs=-1,show_progress_bar=True)

    joblib.dump(study,thismodelpath.joinpath(f'study_{study.study_name}.pkl').as_posix())

    classmodel = init_clasmodel(study.best_trial)
    ytrain=datadf['Actual Rating'].apply(lambda x: x-1)
    classmodel.fit(tf.convert_to_tensor(xtrain),tf.convert_to_tensor(tf.keras.utils.to_categorical(ytrain,num_classes=5)),validation_split=0.1,epochs=10, batch_size=128,verbose=0
    classmodel.save(thismodelpath.joinpath(f'fmodel_clas.h5').as_posix())
else:
    classmodel=tf.keras.models.load_model(thismodelpath.joinpath(f'fmodel_clas.h5').as_posix())
```

```
[I 2024-09-08 17:39:06,429] A new study created in RDB with name: prediction_classification
Directory already exists
  0%|          | 0/20 [00:00<?, ?it/s]
```

```
112/112 [==============================] - 2s 21ms/step - loss: 497.4568 - accuracy: 0.3303
112/112 [==============================] - 2s 20ms/step - loss: 1.3635 - accuracy: 0.3303
112/112 [==============================] - 2s 19ms/step - loss: 1.3625 - accuracy: 0.3525
112/112 [==============================] - 2s 20ms/step - loss: 1.3678 - accuracy: 0.3522
112/112 [==============================] - 2s 19ms/step - loss: 3.5924 - accuracy: 0.2222
112/112 [==============================] - 2s 19ms/step - loss: 1.7471 - accuracy: 0.3068
112/112 [==============================] - 2s 19ms/step - loss: 1.3636 - accuracy: 0.3528
112/112 [==============================] - 3s 28ms/step - loss: 24.7000 - accuracy: 0.2822
112/112 [==============================] - 3s 25ms/step - loss: 1.3619 - accuracy: 0.3528
112/112 [==============================] - 3s 29ms/step - loss: 1.3632 - accuracy: 0.3528
112/112 [==============================] - 3s 23ms/step - loss: 1.3671 - accuracy: 0.3528
112/112 [==============================] - 2s 20ms/step - loss: 1.3667 - accuracy: 0.3303
112/112 [==============================] - 3s 24ms/step - loss: 1.3616 - accuracy: 0.3267
112/112 [==============================] - 2s 21ms/step - loss: 1.6975 - accuracy: 0.1900
112/112 [==============================] - 2s 22ms/step - loss: 1.3625 - accuracy: 0.3528
112/112 [==============================] - 3s 23ms/step - loss: 1.3723 - accuracy: 0.3488
112/112 [==============================] - 2s 16ms/step - loss: 449.9532 - accuracy: 0.3374
112/112 [==============================] - 2s 17ms/step - loss: 1.3745 - accuracy: 0.3397
112/112 [==============================] - 2s 17ms/step - loss: 1.3756 - accuracy: 0.3397
112/112 [==============================] - 2s 19ms/step - loss: 1.7460 - accuracy: 0.3391
112/112 [==============================] - 2s 19ms/step - loss: 1.3752 - accuracy: 0.3397
112/112 [==============================] - 2s 20ms/step - loss: 1.3752 - accuracy: 0.3397
112/112 [==============================] - 2s 21ms/step - loss: 1.3824 - accuracy: 0.3464
112/112 [==============================] - 2s 21ms/step - loss: 1.8635 - accuracy: 0.3170
112/112 [==============================] - 3s 24ms/step - loss: 1.3746 - accuracy: 0.3397
112/112 [==============================] - 3s 22ms/step - loss: 1.3798 - accuracy: 0.3397
112/112 [==============================] - 2s 21ms/step - loss: 1.3746 - accuracy: 0.3293
112/112 [==============================] - 3s 22ms/step - loss: 1.3744 - accuracy: 0.3293
112/112 [==============================] - 3s 23ms/step - loss: 1.3749 - accuracy: 0.3296
112/112 [==============================] - 3s 22ms/step - loss: 1.3741 - accuracy: 0.3397
112/112 [==============================] - 3s 22ms/step - loss: 1.3816 - accuracy: 0.3397
112/112 [==============================] - 2s 21ms/step - loss: 1.3747 - accuracy: 0.3397
112/112 [==============================] - 2s 14ms/step - loss: 115.4966 - accuracy: 0.3548
112/112 [==============================] - 2s 15ms/step - loss: 1.3703 - accuracy: 0.3582
112/112 [==============================] - 2s 15ms/step - loss: 1.3728 - accuracy: 0.3543
112/112 [==============================] - 2s 17ms/step - loss: 5.3304 - accuracy: 0.3464
112/112 [==============================] - 2s 19ms/step - loss: 1.3722 - accuracy: 0.3582
112/112 [==============================] - 2s 20ms/step - loss: 1.3716 - accuracy: 0.3573
112/112 [==============================] - 2s 21ms/step - loss: 1.3714 - accuracy: 0.3582
112/112 [==============================] - 2s 19ms/step - loss: 1.4665 - accuracy: 0.3383
112/112 [==============================] - 2s 21ms/step - loss: 1.3712 - accuracy: 0.3582
112/112 [==============================] - 2s 21ms/step - loss: 1.3710 - accuracy: 0.3582
112/112 [==============================] - 2s 20ms/step - loss: 1.3709 - accuracy: 0.3582
112/112 [==============================] - 2s 21ms/step - loss: 1.3705 - accuracy: 0.3582
112/112 [==============================] - 2s 20ms/step - loss: 1.3726 - accuracy: 0.3573
112/112 [==============================] - 3s 22ms/step - loss: 1.3713 - accuracy: 0.3582
112/112 [==============================] - 2s 21ms/step - loss: 1.3752 - accuracy: 0.3206
112/112 [==============================] - 2s 19ms/step - loss: 1.3709 - accuracy: 0.3582
112/112 [==============================] - 2s 16ms/step - loss: 308.9193 - accuracy: 0.2351
112/112 [==============================] - 2s 14ms/step - loss: 1.3647 - accuracy: 0.3192
112/112 [==============================] - 2s 14ms/step - loss: 1.3642 - accuracy: 0.3590
112/112 [==============================] - 2s 16ms/step - loss: 1.4059 - accuracy: 0.3133
112/112 [==============================] - 2s 18ms/step - loss: 1.3661 - accuracy: 0.3192
112/112 [==============================] - 2s 17ms/step - loss: 1.3629 - accuracy: 0.3509
112/112 [==============================] - 2s 18ms/step - loss: 1.4386 - accuracy: 0.3246
112/112 [==============================] - 2s 20ms/step - loss: 1.3620 - accuracy: 0.3593
112/112 [==============================] - 2s 20ms/step - loss: 1.3622 - accuracy: 0.3593
112/112 [==============================] - 2s 20ms/step - loss: 1.3627 - accuracy: 0.3593
112/112 [==============================] - 2s 21ms/step - loss: 1.3619 - accuracy: 0.3593
112/112 [==============================] - 2s 17ms/step - loss: 1.3637 - accuracy: 0.3593
112/112 [==============================] - 2s 20ms/step - loss: 1.3623 - accuracy: 0.3593
112/112 [==============================] - 2s 20ms/step - loss: 1.3623 - accuracy: 0.3515
112/112 [==============================] - 2s 20ms/step - loss: 1.3635 - accuracy: 0.3593
112/112 [==============================] - 2s 19ms/step - loss: 67.7628 - accuracy: 0.2912
 45/112 [===========>..................] - ETA: 1s - loss: 1.4411 - accuracy: 0.3139

 99/112 [=========================>....] - ETA: 0s - loss: 1.3639 - accuracy: 0.3586[I 2024-09-08 17:40:59,366] Trial 6 finished with value: 0.3097909063100815 and parameters:
{'n_layers': 1, 'activation': 'relu', 'n_units_l0': 90}. Best is trial 6 with value: 0.3097909063100815.
  0%|          | 0/20 [01:52<?, ?it/s]
108/112 [===========================>..] - ETA: 0s - loss: 1.3672 - accuracy: 0.3582
Best trial: 6. Best value: 0.309791:   5%|▌         | 1/20 [01:53<35:50, 113.17s/it]
112/112 [==============================] - 2s 20ms/step - loss: 1.3631 - accuracy: 0.3593
112/112 [==============================] - 2s 19ms/step - loss: 1.3793 - accuracy: 0.3498
Best trial: 2. Best value: 0.339443:   5%|▌         | 1/20 [01:54<35:50, 113.17s/it]
[I 2024-09-08 17:41:00,991] Trial 2 finished with value: 0.3394433677196503 and parameters: {'n_layers': 2, 'activation': 'sigmoid', 'n_units_l0': 65, 'n_units_l1': 105}. Best is
trial 2 with value: 0.3394433677196503.
Best trial: 2. Best value: 0.339443:  10%|▉         | 2/20 [01:54<14:15, 47.55s/it]
112/112 [==============================] - 2s 20ms/step - loss: 1.3912 - accuracy: 0.3156
Best trial: 2. Best value: 0.339443:  10%|▉         | 2/20 [02:00<14:15, 47.55s/it]
[I 2024-09-08 17:41:06,491] Trial 9 finished with value: 0.3441506624221802 and parameters: {'n_layers': 1, 'activation': 'tanh', 'n_units_l0': 63}. Best is trial 9 with value:
0.3441506624221802.
Best trial: 9. Best value: 0.344151:  15%|█▍        | 3/20 [02:00<08:02, 28.37s/it]
112/112 [==============================] - 2s 18ms/step - loss: 1.3944 - accuracy: 0.3338
 28/112 [======>.......................] - ETA: 1s - loss: 1.3482 - accuracy: 0.3237

 17/112 [===>..........................] - ETA: 1s - loss: 1.3549 - accuracy: 0.3382
Best trial: 9. Best value: 0.344151:  15%|█▍        | 3/20 [02:02<08:02, 28.37s/it]
[I 2024-09-08 17:41:09,201] Trial 1 finished with value: 0.31097409725189207 and parameters: {'n_layers': 5, 'activation': 'relu', 'n_units_l0': 68, 'n_units_l1': 109, 'n_units_l
2': 58, 'n_units_l3': 35, 'n_units_l4': 67}. Best is trial 9 with value: 0.3441506624221802.
 24/112 [=====>........................] - ETA: 1s - loss: 1.3639 - accuracy: 0.3333
Best trial: 9. Best value: 0.344151:  20%|█▉        | 4/20 [02:03<04:51, 18.23s/it]
112/112 [==============================] - 2s 22ms/step - loss: 1.3814 - accuracy: 0.3153
112/112 [==============================] - 3s 23ms/step - loss: 1.3815 - accuracy: 0.3442
 16/112 [===>..........................] - ETA: 2s - loss: 1.3501 - accuracy: 0.3418

 61/112 [===============>..............] - ETA: 1s - loss: 1.4089 - accuracy: 0.3345
 18/112 [===>..........................] - ETA: 2s - loss: 1.3575 - accuracy: 0.3420[I 2024-09-08 17:41:12,042] Trial 13 finished with value: 0.33686363101005556 and parameters:
{'n_layers': 1, 'activation': 'sigmoid', 'n_units_l0': 107}. Best is trial 7 with value: 0.3490833818912506.
Best trial: 9. Best value: 0.344151:  20%|█▉        | 4/20 [02:05<04:51, 18.23s/it]
 21/112 [====>.......................] - ETA: 2s - loss: 1.3409 - accuracy: 0.3408

 65/112 [================>.............] - ETA: 1s - loss: 1.4028 - accuracy: 0.3380
 23/112 [=====>........................] - ETA: 2s - loss: 1.3502 - accuracy: 0.3383[I 2024-09-08 17:41:12,098] Trial 7 finished with value: 0.3490833818912506 and parameters:
{'n_layers': 5, 'activation': 'tanh', 'n_units_l0': 34, 'n_units_l1': 34, 'n_units_l2': 111, 'n_units_l3': 62, 'n_units_l4': 65}. Best is trial 7 with value: 0.3490833818912506.
Best trial: 9. Best value: 0.344151:  20%|█▉        | 4/20 [02:05<04:51, 18.23s/it]
 29/112 [======>.......................] - ETA: 2s - loss: 1.3491 - accuracy: 0.3491
Best trial: 7. Best value: 0.349083:  25%|██▍       | 5/20 [02:06<03:11, 12.76s/it]
 74/112 [==================>...........] - ETA: 0s - loss: 1.4074 - accuracy: 0.3378
Best trial: 7. Best value: 0.349083:  25%|██▍       | 5/20 [02:06<03:11, 12.76s/it]
112/112 [==============================] - 3s 27ms/step - loss: 1.3868 - accuracy: 0.3428
 87/112 [======================>.......] - ETA: 0s - loss: 1.3948 - accuracy: 0.3369

 90/112 [=======================>......] - ETA: 0s - loss: 1.3896 - accuracy: 0.3375
```

```
 54/112 [=============>.............] - ETA: 1s - loss: 1.4262 - accuracy: 0.3316[I 2024-09-08 17:41:14,222] Trial 14 finished with value: 0.32094831466674806 and parameters:
{'n_layers': 4, 'activation': 'relu', 'n_units_l0': 44, 'n_units_l1': 128, 'n_units_l2': 75, 'n_units_l3': 90}. Best is trial 7 with value: 0.3490833818912506.
 23/112 [=====>........................] - ETA: 2s - loss: 1.3486 - accuracy: 0.3383
Best trial: 7. Best value: 0.349083:  30%|███       | 6/20 [02:07<02:58, 12.76s/it]
 33/112 [=======>......................] - ETA: 2s - loss: 1.3545 - accuracy: 0.3466
Best trial: 7. Best value: 0.349083:  30%|███       | 6/20 [02:08<02:58, 12.76s/it]
 42/112 [==========>...................] - ETA: 1s - loss: 1.4217 - accuracy: 0.3244
Best trial: 7. Best value: 0.349083:  35%|███       | 7/20 [02:08<01:28,  6.84s/it]
112/112 [==============================] - 3s 31ms/step - loss: 1.3805 - accuracy: 0.3442
 32/112 [=======>......................] - ETA: 2s - loss: 1.3480 - accuracy: 0.3477
Best trial: 7. Best value: 0.349083:  35%|███       | 7/20 [02:09<01:28,  6.84s/it]
[I 2024-09-08 17:41:15,491] Trial 3 finished with value: 0.3508210301399231 and parameters: {'n_layers': 5, 'activation': 'sigmoid', 'n_units_l0': 47, 'n_units_l1': 72, 'n_units_
l2': 93, 'n_units_l3': 85, 'n_units_l4': 37}. Best is trial 3 with value: 0.3508210301399231.
102/112 [==========================>...] - ETA: 0s - loss: 1.3868 - accuracy: 0.3422
Best trial: 3. Best value: 0.350821:  35%|███       | 7/20 [02:09<01:28,  6.84s/it]
 92/112 [=======================>......] - ETA: 0s - loss: 1.3929 - accuracy: 0.3370
Best trial: 3. Best value: 0.350821:  40%|████      | 8/20 [02:09<01:04,  5.34s/it]
112/112 [==============================] - 3s 29ms/step - loss: 1.3806 - accuracy: 0.3442
112/112 [==============================] - 3s 25ms/step - loss: 1.3805 - accuracy: 0.3442
105/112 [===========================>..] - ETA: 0s - loss: 1.3851 - accuracy: 0.3438

  1/112 [..............................] - ETA: 6s - loss: 1.1377 - accuracy: 0.4375
Best trial: 3. Best value: 0.350821:  40%|████      | 8/20 [02:10<01:04,  5.34s/it]
[I 2024-09-08 17:41:16,478] Trial 4 finished with value: 0.3429760754108429 and parameters: {'n_layers': 8, 'activation': 'relu', 'n_units_l0': 96, 'n_units_l1': 72, 'n_units_l
2': 126, 'n_units_l3': 44, 'n_units_l4': 120, 'n_units_l5': 105, 'n_units_l6': 41, 'n_units_l7': 48}. Best is trial 3 with value: 0.3508210301399231.
112/112 [==============================] - 3s 26ms/step - loss: 1.3804 - accuracy: 0.3442

 80/112 [====================>.........] - ETA: 0s - loss: 1.3930 - accuracy: 0.3383[I 2024-09-08 17:41:16,632] Trial 8 finished with value: 0.3442639946937561 and parameters:
{'n_layers': 6, 'activation': 'sigmoid', 'n_units_l0': 91, 'n_units_l1': 118, 'n_units_l2': 87, 'n_units_l3': 86, 'n_units_l4': 51, 'n_units_l5': 63}. Best is trial 3 with value:
0.3508210301399231.
  9/112 [=>............................] - ETA: 2s - loss: 1.2900 - accuracy: 0.3542
Best trial: 3. Best value: 0.350821:  40%|████      | 8/20 [02:10<01:04,  5.34s/it]
 84/112 [=====================>........] - ETA: 0s - loss: 1.3918 - accuracy: 0.3378
Best trial: 3. Best value: 0.350821:  40%|████      | 8/20 [02:10<01:04,  5.34s/it]
 16/112 [===>..........................] - ETA: 2s - loss: 1.3495 - accuracy: 0.3418
Best trial: 3. Best value: 0.350821:  45%|████      | 9/20 [02:10<00:45,  4.12s/it]
 88/112 [======================>.......] - ETA: 0s - loss: 1.3938 - accuracy: 0.3366
Best trial: 3. Best value: 0.350821:  45%|████      | 9/20 [02:10<00:45,  4.12s/it]
 20/112 [====>.........................] - ETA: 2s - loss: 1.3470 - accuracy: 0.3375
Best trial: 3. Best value: 0.350821:  50%|████      | 10/20 [02:10<00:29,  3.00s/it]
 93/112 [=======================>......] - ETA: 0s - loss: 1.3905 - accuracy: 0.3380

 23/112 [=====>........................] - ETA: 2s - loss: 1.3512 - accuracy: 0.3383
Best trial: 3. Best value: 0.350821:  50%|████      | 10/20 [02:10<00:29,  3.00s/it]
[I 2024-09-08 17:41:17,076] Trial 15 finished with value: 0.3508210301399231 and parameters: {'n_layers': 5, 'activation': 'tanh', 'n_units_l0': 53, 'n_units_l1': 52, 'n_units_l
2': 39, 'n_units_l3': 56, 'n_units_l4': 126}. Best is trial 3 with value: 0.3508210301399231.
 39/112 [=========>....................] - ETA: 1s - loss: 1.3979 - accuracy: 0.3325
Best trial: 3. Best value: 0.350821:  50%|████      | 10/20 [02:10<00:29,  3.00s/it]
109/112 [============================>.] - ETA: 0s - loss: 1.3814 - accuracy: 0.3449
Best trial: 3. Best value: 0.350821:  55%|████      | 11/20 [02:10<00:20,  2.26s/it]
112/112 [==============================] - 3s 26ms/step - loss: 1.3804 - accuracy: 0.3442
 63/112 [===============>..............] - ETA: 0s - loss: 1.3961 - accuracy: 0.3388
Best trial: 3. Best value: 0.350821:  55%|████      | 11/20 [02:11<00:20,  2.26s/it]
[I 2024-09-08 17:41:17,752] Trial 11 finished with value: 0.3487470388412476 and parameters: {'n_layers': 6, 'activation': 'sigmoid', 'n_units_l0': 44, 'n_units_l1': 93, 'n_units
_l2': 51, 'n_units_l3': 93, 'n_units_l4': 108, 'n_units_l5': 33}. Best is trial 3 with value: 0.3508210301399231.
 74/112 [==================>...........] - ETA: 0s - loss: 1.3992 - accuracy: 0.3404
Best trial: 3. Best value: 0.350821:  55%|████      | 11/20 [02:11<00:20,  2.26s/it]
 78/112 [===================>..........] - ETA: 0s - loss: 1.3932 - accuracy: 0.3401
Best trial: 3. Best value: 0.350821:  60%|████      | 12/20 [02:11<00:14,  1.80s/it]
112/112 [==============================] - 2s 16ms/step - loss: 1.3806 - accuracy: 0.3442
Best trial: 3. Best value: 0.350821:  60%|████      | 12/20 [02:12<00:14,  1.80s/it]
[I 2024-09-08 17:41:18,578] Trial 12 finished with value: 0.31607679128864685 and parameters: {'n_layers': 6, 'activation': 'relu', 'n_units_l0': 33, 'n_units_l1': 116, 'n_units_l
2': 47, 'n_units_l3': 88, 'n_units_l4': 106, 'n_units_l5': 36}. Best is trial 3 with value: 0.3508210301399231.
  6/112 [>.............................] - ETA: 1s - loss: 1.2859 - accuracy: 0.3542
Best trial: 3. Best value: 0.350821:  65%|████      | 13/20 [02:12<00:10,  1.46s/it]
112/112 [==============================] - 1s 8ms/step - loss: 1.3809 - accuracy: 0.3442
112/112 [==============================] - 1s 9ms/step - loss: 1.3826 - accuracy: 0.3442
  1/112 [..............................] - ETA: 4s - loss: 1.1466 - accuracy: 0.4375
Best trial: 3. Best value: 0.350821:  65%|████      | 13/20 [02:13<00:10,  1.46s/it]
[I 2024-09-08 17:41:19,774] Trial 10 finished with value: 0.3440399765968323 and parameters: {'n_layers': 8, 'activation': 'tanh', 'n_units_l0': 71, 'n_units_l1': 37, 'n_units_l
2': 120, 'n_units_l3': 93, 'n_units_l4': 116, 'n_units_l5': 91, 'n_units_l6': 123, 'n_units_l7': 43}. Best is trial 3 with value: 0.3508210301399231.
  4/112 [>.............................] - ETA: 1s - loss: 1.2328 - accuracy: 0.3594
Best trial: 3. Best value: 0.350821:  65%|████      | 13/20 [02:13<00:10,  1.46s/it]
[I 2024-09-08 17:41:19,831] Trial 0 finished with value: 0.3433098196983337 and parameters: {'n_layers': 9, 'activation': 'sigmoid', 'n_units_l0': 49, 'n_units_l1': 49, 'n_units
_l2': 96, 'n_units_l3': 36, 'n_units_l4': 84, 'n_units_l5': 33, 'n_units_l6': 34, 'n_units_l7': 81, 'n_units_l8': 127}. Best is trial 3 with value: 0.3508210301399231.
  6/112 [>.............................] - ETA: 2s - loss: 1.2867 - accuracy: 0.3542
Best trial: 3. Best value: 0.350821:  65%|████      | 13/20 [02:13<00:10,  1.46s/it]
  9/112 [=>............................] - ETA: 2s - loss: 1.2906 - accuracy: 0.3542
Best trial: 3. Best value: 0.350821:  70%|████      | 14/20 [02:13<00:08,  1.40s/it]
112/112 [==============================] - 1s 5ms/step - loss: 1.3795 - accuracy: 0.3442
Best trial: 3. Best value: 0.350821:  75%|████      | 15/20 [02:14<00:06,  1.40s/it]
[I 2024-09-08 17:41:20,431] Trial 5 finished with value: 0.3500364959523996 and parameters: {'n_layers': 10, 'activation': 'relu', 'n_units_l0': 59, 'n_units_l1': 45, 'n_units_l
2': 69, 'n_units_l3': 70, 'n_units_l4': 71, 'n_units_l5': 112, 'n_units_l6': 45, 'n_units_l7': 114, 'n_units_l8': 47, 'n_units_l9': 71}. Best is trial 3 with value: 0.35082103013
99231.
Best trial: 3. Best value: 0.350821:  80%|████      | 16/20 [02:14<00:03,  1.14it/s]
112/112 [==============================] - 0s 2ms/step - loss: 1.3654 - accuracy: 0.3533
112/112 [==============================] - 0s 3ms/step - loss: 1.3644 - accuracy: 0.3525
112/112 [==============================] - 0s 3ms/step - loss: 1.3652 - accuracy: 0.3303
112/112 [==============================] - 0s 2ms/step - loss: 3.2718 - accuracy: 0.3516
112/112 [==============================] - 0s 2ms/step - loss: 1.3747 - accuracy: 0.3397
112/112 [==============================] - 0s 2ms/step - loss: 1.3766 - accuracy: 0.3422
112/112 [==============================] - 0s 2ms/step - loss: 1.3747 - accuracy: 0.3397
112/112 [==============================] - 0s 2ms/step - loss: 1.4446 - accuracy: 0.3184
112/112 [==============================] - 0s 2ms/step - loss: 1.3712 - accuracy: 0.3582
112/112 [==============================] - 0s 2ms/step - loss: 1.3726 - accuracy: 0.3293
112/112 [==============================] - 0s 3ms/step - loss: 1.3713 - accuracy: 0.3582
112/112 [==============================] - 0s 3ms/step - loss: 1.5017 - accuracy: 0.3582
112/112 [==============================] - 0s 3ms/step - loss: 1.3721 - accuracy: 0.3192
112/112 [==============================] - 0s 2ms/step - loss: 1.3651 - accuracy: 0.3587
112/112 [==============================] - 0s 2ms/step - loss: 1.3635 - accuracy: 0.3192
112/112 [==============================] - 0s 3ms/step - loss: 1.3803 - accuracy: 0.3442
  1/112 [..............................] - ETA: 3s - loss: 1.2406 - accuracy: 0.4375
Best trial: 3. Best value: 0.350821:  80%|████      | 16/20 [02:31<00:03,  1.14it/s]
[I 2024-09-08 17:41:37,428] Trial 17 finished with value: 0.34291741251945496 and parameters: {'n_layers': 3, 'activation': 'tanh', 'n_units_l0': 83, 'n_units_l1': 32, 'n_units_l
2': 73}. Best is trial 3 with value: 0.3508210301399231.
Best trial: 3. Best value: 0.350821:  85%|████      | 17/20 [02:31<00:14,  4.85s/it]
112/112 [==============================] - 0s 3ms/step - loss: 1.3890 - accuracy: 0.3341
112/112 [==============================] - 0s 2ms/step - loss: 1.3795 - accuracy: 0.3456
Best trial: 3. Best value: 0.350821:  85%|████      | 17/20 [02:31<00:14,  4.85s/it]
[I 2024-09-08 17:41:37,956] Trial 19 finished with value: 0.34566410183906554 and parameters: {'n_layers': 2, 'activation': 'tanh', 'n_units_l0': 37, 'n_units_l1': 38}. Best is t
rial 3 with value: 0.3508210301399231.
```

```
Best trial: 3. Best value: 0.350821:  90%|████████ | 18/20 [02:31<00:07,  3.72s/it]
112/112 [==============================] - 0s 1ms/step - loss: 1.3814 - accuracy: 0.3442
Best trial: 3. Best value: 0.350821:  95%|████████ | 19/20 [02:33<00:03,  3.20s/it]
[I 2024-09-08 17:41:39,753] Trial 16 finished with value: 0.33832228779792783 and parameters: {'n_layers': 5, 'activation': 'sigmoid', 'n_units_l0': 64, 'n_units_l1': 62, 'n_units_l2': 45, 'n_units_l3': 100, 'n_units_l4': 98}. Best is trial 3 with value: 0.3508210301399231.
112/112 [==============================] - 0s 1ms/step - loss: 1.3884 - accuracy: 0.3430
Best trial: 3. Best value: 0.350821: 100%|████████| 20/20 [02:34<00:00,  7.72s/it]
[I 2024-09-08 17:41:40,820] Trial 18 finished with value: 0.34106773734092 71 and parameters: {'n_layers': 7, 'activation': 'relu', 'n_units_l0': 40, 'n_units_l1': 94, 'n_units_l2': 77, 'n_units_l3': 55, 'n_units_l4': 41, 'n_units_l5': 49, 'n_units_l6': 35}. Best is trial 3 with value: 0.3508210301399231.
```

Compare models to Predicted Rating

In [111…

```
datadf=pd.read_csv(datawreviewsfile,sep=';')
# fulldatadf=pd.read_csv(dataworeviewsfile,sep=';')
datadf=datadf.drop(['biography', 'children', 'comics', 'crime', 'fantasy', 'fiction',
       'graphic', 'historicalfiction', 'history', 'mystery', 'nonfiction',
       'paranormal', 'poetry', 'romance', 'thriller', 'youngadult'],axis=1)

def remove_punctuation(x):
    cleanedtext=re.sub('[^A-Za-z0-9]+', ' ', x)
    return cleanedtext
datadf['description']=datadf.description.transform(remove_punctuation)
datadf['genre']=datadf.genre.apply(lambda x: x.replace("'",")) # remove '
datadf['genre']=datadf.genre.apply(lambda x: x.replace("-",")) # revome -
datadf['genre']=datadf.genre.apply(lambda x: x.replace(" ",")) # remove spaces
datadf['genre']=datadf.genre.apply(lambda x: x[1:-1].split(',')) # split into list

mlbmodelpath=models_path.joinpath('mlb_model')
mlbmodel=joblib.load(mlbmodelpath.joinpath('mlbmodel.pkl').as_posix())
newcols=mlbmodel.classes_
pred=mlbmodel.transform(datadf.genre)
preddf=pd.DataFrame(pred,columns=newcols)
datadf=datadf.join(preddf)

descriptionmodelpath=models_path.joinpath('description_models')
descmodel=joblib.load(descriptionmodelpath.joinpath('gridamodel.pkl').as_posix())
newcols=descmodel.best_estimator_.named_steps['lda'].get_feature_names_out()
pred=descmodel.best_estimator_.transform(datadf.description)
preddf=pd.DataFrame(pred,columns=newcols)
datadf=datadf.join(preddf)

x=datadf[['user_id_mapping', 'num_pages',
       'ratings_count', 'biography', 'children', 'comics', 'crime', 'fantasy',
       'fiction', 'graphic', 'historicalfiction', 'history', 'mystery',
       'nonfiction', 'paranormal', 'poetry', 'romance', 'thriller',
       'youngadult', 'latentdirichletallocation0',
       'latentdirichletallocation1', 'latentdirichletallocation2']]
y=datadf[['Predicted Rating', 'Actual Rating',]]

def get_class_pred(x):
    pred=classmodel.predict(tf.convert_to_tensor(x))
    classpred=[]
    for x in pred:
        classpred.append(np.argmax(x)+1)
    return classpred
```

In [112…

```
y['regsvm']=regsvmmodel.best_estimator_.predict(x)
y['regdt']=regdtmodel.best_estimator_.predict(x)
y['classvm']=classvmmodel.best_estimator_.predict(x)
y['clasdt']=clasdtmodel.best_estimator_.predict(x)
y['regmodel']=regmodel.predict(tf.convert_to_tensor(x))
y['clasmodel']=get_class_pred(x)
```

```
620/620 [==============================] - 1s 726us/step
620/620 [==============================] - 1s 920us/step
```

In [113…

```
baseline=mean_squared_error(y['Actual Rating'],y['Predicted Rating'])
regsvmscore=mean_squared_error(y['Actual Rating'],y['regsvm'])
regtdtscore=mean_squared_error(y['Actual Rating'],y['regdt'])
classvmscore=mean_squared_error(y['Actual Rating'],y['classvm'])
clasdtscore=mean_squared_error(y['Actual Rating'],y['clasdt'])
clasmodelscore=mean_squared_error(y['Actual Rating'],y['clasmodel'])
regmodelscore=mean_squared_error(y['Actual Rating'],y['regmodel'])
```

In [114…

```
# Print Mean Squared Error for each model
# --> some may only predicts 5*s, not robust models
print(f'MSE Predicted Rating: {baseline}')
print(f'MSE regsvm: {regsvmscore}')
print(f'MSE regdt: {regtdtscore}')
print(f'MSE classsvm: {classvmscore}')
print(f'MSE clasdt: {clasdtscore}')
print(f'MSE clasmodel: {clasmodelscore}')
print(f'MSE regmodel: {regmodelscore}')
```

```
MSE Predicted Rating: 0.7844668079096044
MSE regsvm: 1.3151504084807035
MSE regdt: 1.134421246139446
MSE classsvm: 2.0202784503631963
MSE clasdt: 1.4445621468926553
MSE clasmodel: 1.1058313155770783
MSE regmodel: 1.0991262770282197
```

- Comparing the trained models to the Predicted Rating included in the dataset, it is obvious that the model quality is lower than the one used by the dataset creator.
- Regression Trees seem to come the closest to the Predicted Rating (In previous runs (MSE: 0.9), not true anymore. Regression Neural Network is better now).
- SVR und SVM had major problems during training and weren't converging correctly. Scalers were suggested in the warnings and might have helped those issues. Time constraints did not allow for more fiddling with optimization.
- On average the prediction made by Regression models seems to have performed better. In the case of neural networks, the predictions seem of similar quality, so the path easiest to optimize and implement should be further optimized.
- The trained models include the user id of the reviewer as an input feature. This was determined to be unusable from the beginning, so confidence in the models was not high from the beginning. The user id was supposed to be exchanged for a representation of the users reading preferences created via clustering. As the clustering approaches proved non-conclusive in the time given, no replacement took place. Further analysis regarding representing the user by the books he enjoyed and didn't enjoy should take place. Averaging scores by genre or fuzzy clustering approaches could be evaluated.
- The reviews were unbalanced towards better scores, so boosting ensembles made from Regression models could prove more accurate
- It was determied that quite a bit more work could be put into preprocessing the data to observe the impact on the rating prediction
- A deep dive into the hyperparameters of models trained could prove helpful

Prediction based on information about users that have rated

Starting with a more detailed analysis of the merged data set from the experiment to cluster users.

In [115…
```
# inspired by Leas approach to analyze the initial dataset

print(f"Number of observations: {len(merged_df)}")
print(f"Number of unique book ids: {len(merged_df['book_id_x'].unique())}")
print(f"Number of unique user ids: {len(merged_df['user_id_mapping'].unique())}")


#limiting the output to 2 decimal digits for average rating
print(f"Average actual rating: {merged_df['Actual Rating'].mean():.2f}")
print(f"Average predicted rating: {merged_df['Predicted Rating'].mean():.2f}")
```

```
Number of observations: 19824
Number of unique book ids: 95
Number of unique user ids: 17103
Average actual rating: 3.87
Average predicted rating: 3.86
```

In [116…
```
# max numnber of ratings per user and others
# groupby user id
merged_df_userids=merged_df[["user_id_mapping","book_id_x"]].groupby(by="user_id_mapping").count()
merged_df_userids=merged_df_userids.reset_index()
merged_df_userids=merged_df_userids.rename(columns={"book_id_x":"book_count"})
```

In [117…
```
print(f"(repeat) Number of users: {len(merged_df_userids)}")
print(f"Max number of books revied per user: {merged_df_userids.book_count.max()}")
print(f"Min number of books revied per user: {merged_df_userids.book_count.min()}")
print(f"Number of users with 1 review: {len(merged_df_userids.query('book_count == 1'))}")
print(f"... in %: {round(len(merged_df_userids.query('book_count == 1'))/len(merged_df_userids),2)*100}%")
print(f"Number of users with 2 review: {len(merged_df_userids.query('book_count == 2'))}")
print(f"... in %: {round(len(merged_df_userids.query('book_count == 2'))/len(merged_df_userids),2)*100}%")
print(f"Number of users with less than 3 reviews: {len(merged_df_userids.query('book_count < 3'))}")
print(f"... in %: {round(len(merged_df_userids.query('book_count < 3'))/len(merged_df_userids),2)*100}%")
print(f"Number of users with more than 1 review: {len(merged_df_userids.query('book_count >1'))}")
print(f"... in %: {len(merged_df_userids.query('book_count > 1')) / len(merged_df_userids) * 100:.2f}%")
```

```
(repeat) Number of users: 17103
Max number of books revied per user: 5
Min number of books revied per user: 1
Number of users with 1 review: 14668
... in %: 86.0%
Number of users with 2 review: 2181
... in %: 13.0%
Number of users with less than 3 reviews: 16849
... in %: 99.0%
Number of users with more than 1 review: 2435
... in %: 14.24%
```

In [118…
```
# group by the user IDs and count the entries per userid
id_counts = merged_df.groupby('user_id_mapping').size()
print(id_counts.head())
```

```
user_id_mapping
2     1
4     1
7     1
21    1
23    1
dtype: int64
```

For the prediction, use only the data of users that have 2 or more book ratings.

In [119…
```
# filter for user-ids with 2 or more entries

ids_with_multiple_entries = id_counts[id_counts >= 2].index

# Filter the original DataFrame to only include these IDs
df_filtered = merged_df[merged_df['user_id_mapping'].isin(ids_with_multiple_entries)]
print(ids_with_multiple_entries.shape)
print(df_filtered.shape)
```

```
(2435,)
(5156, 39)
```

In [120…
```
# more analyis
# max numnber of ratings per user and others
# groupby user id

df_filtered_userids=df_filtered[["user_id_mapping","book_id_x"]].groupby(by="user_id_mapping").count()
df_filtered_userids=df_filtered_userids.reset_index()
df_filtered_userids=df_filtered_userids.rename(columns={"book_id_x":"book_count"})
```

In [121…
```
#verifiying that now we have a dataframe where every user has reviewed at least 2 books
print(f"(repeat) Number of users: {len(df_filtered_userids)}")
print(f"Max number of books revied per user: {df_filtered_userids.book_count.max()}")
print(f"Min number of books revied per user: {df_filtered_userids.book_count.min()}")
```

```
(repeat) Number of users: 2435
Max number of books revied per user: 5
Min number of books revied per user: 2
```

A dataframe with 5156 lines with entries of 2435 users that have at least 2 ratings will be used for the predictions

In [122…
```
#Show information about the merged and grouped dataset

print(f"Number of users with more than 1 review: {len(df_filtered_userids.query('book_count >1'))}")
print(df_filtered_userids.columns)
print(df_filtered_userids.shape)
print(df_filtered.shape)
```

```
Number of users with more than 1 review: 2435
Index(['user_id_mapping', 'book_count'], dtype='object')
(2435, 2)
(5156, 39)
```

In [123…
```
print(df_filtered.columns)
```

```
Index(['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping',
       'Predicted Rating', 'Actual Rating', 'book_id_y', 'title_y',
       'image_url', 'url', 'num_pages', 'ratings_count', 'description', 'name',
       ' biography', ' children', ' comics', ' crime', ' fantasy', ' fiction',
       ' graphic', ' historical fiction', ' history', ' mystery',
       ' nonfiction', ' paranormal', ' poetry', ' romance', ' thriller',
       ' youngadult', 'children', 'comics', 'fantasy', 'fiction', 'mystery',
       'nonfiction', 'poetry', 'romance', 'youngadult'],
      dtype='object')
```

In [124…
```
#remove columns that are not needed going forward, e.g with double entry and text only, that cannot be used in the algorithm
df_prediction = df_filtered.drop(['Predicted Rating', 'book_id_y', 'title_y', 'description', 'image_url', 'url', 'name'],axis=1 )
```

In [125…
```
#verify that  code did what it should do
print(df_prediction.columns)
```

```
Index(['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping',
       'Actual Rating', 'num_pages', 'ratings_count', ' biography',
       ' children', ' comics', ' crime', ' fantasy', ' fiction', ' graphic',
       ' historical fiction', ' history', ' mystery', ' nonfiction',
       ' paranormal', ' poetry', ' romance', ' thriller', ' youngadult',
       'children', 'comics', 'fantasy', 'fiction', 'mystery', 'nonfiction',
       'poetry', 'romance', 'youngadult'],
      dtype='object')
```

In [126…
```
print(df_prediction.head())
print(df_prediction.shape)
```

```
                    title_x  book_id_x  user_id_mapping  book_id_mapping  \
1   We Should All Be Feminists   22738563            45548              873
4   We Should All Be Feminists   22738563            41888              873
10  We Should All Be Feminists   22738563            19426              873
13  We Should All Be Feminists   22738563            46744              873
14  We Should All Be Feminists   22738563            26977              873

    Actual Rating  num_pages  ratings_count   biography   children   comics  \
1               4       1007         151473           0          0        0
4               5       1007         151473           0          0        0
10              5       1007         151473           0          0        0
13              5       1007         151473           0          0        0
14              4       1007         151473           0          0        0

    ...   youngadult  children  comics  fantasy  fiction  mystery  nonfiction  \
1   ...            0         0       0        1        0        0           0
4   ...            0         0       0        1        0        0           0
10  ...            0         0       0        1        0        0           0
13  ...            0         0       0        1        0        0           0
14  ...            0         0       0        1        0        0           0

    poetry  romance  youngadult
1        0        0           0
4        0        0           0
10       0        0           0
13       0        0           0
14       0        0           0

[5 rows x 32 columns]
(5156, 32)
```

Before going into the prediction, based on what was discussed during the lecture on site in Hamburg, more visualization is prepared to help understand the data.

In [127…
```
# making the columns to feature to prepare prediction
# starting with all columns in the df are features
X = df_prediction.copy()  # Copy the DataFrame to preserve the original
print(X)
```

```
                           title_x  book_id_x  user_id_mapping  \
1          We Should All Be Feminists   22738563            45548
4          We Should All Be Feminists   22738563            41888
10         We Should All Be Feminists   22738563            19426
13         We Should All Be Feminists   22738563            46744
14         We Should All Be Feminists   22738563            26977
...                              ...        ...              ...
19816  The Wonderful Wizard of Oz Oz 1     236093             6263
19817  The Wonderful Wizard of Oz Oz 1     236093            27954
19818  The Wonderful Wizard of Oz Oz 1     236093            24286
19819  The Wonderful Wizard of Oz Oz 1     236093            16631
19822  The Wonderful Wizard of Oz Oz 1     236093             6029

       book_id_mapping  Actual Rating  num_pages  ratings_count   biography  \
1                  873              4       1007         151473           0
4                  873              5       1007         151473           0
10                 873              5       1007         151473           0
13                 873              5       1007         151473           0
14                 873              4       1007         151473           0
...                ...            ...        ...            ...         ...
19816               19              5        178         147305           0
19817               19              4        178         147305           0
19818               19              4        178         147305           0
19819               19              4        178         147305           0
19822               19              3        178         147305           0

        children   comics  ...   youngadult  children  comics  fantasy  \
1              0       0  ...            0         0       0        1
4              0       0  ...            0         0       0        1
10             0       0  ...            0         0       0        1
13             0       0  ...            0         0       0        1
14             0       0  ...            0         0       0        1
...          ...     ...  ...          ...       ...     ...      ...
19816          0       0  ...            0         0       0        0
19817          0       0  ...            0         0       0        0
19818          0       0  ...            0         0       0        0
19819          0       0  ...            0         0       0        0
19822          0       0  ...            0         0       0        0

       fiction  mystery  nonfiction  poetry  romance  youngadult
1            0        0           0       0        0           0
4            0        0           0       0        0           0
10           0        0           0       0        0           0
13           0        0           0       0        0           0
14           0        0           0       0        0           0
...        ...      ...         ...     ...      ...         ...
19816        0        0           0       0        0           1
19817        0        0           0       0        0           1
19818        0        0           0       0        0           1
19819        0        0           0       0        0           1
19822        0        0           0       0        0           1

[5156 rows x 32 columns]
```

In [128…
```
sns.set(font_scale=1)
sns.set_style('whitegrid')
```

In [129…
```
# Access the feature names
feature_names = df_prediction.columns.tolist()
print("Feature names:", feature_names)
```

```
Feature names: ['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping', 'Actual Rating', 'num_pages', 'ratings_count', ' biography', ' children', ' comics', ' crime', ' fantasy', ' fiction', ' graphic', ' historical fiction', ' history', ' mystery', ' nonfiction', ' paranormal', ' poetry', ' romance', ' thriller', ' youngadult', 'children', 'comics', 'fantasy', 'fiction', 'mystery', 'nonfiction', 'poetry', 'romance', 'youngadult']
```

In [130…
```
# Adding the feature_names as an attribute to the DataFrame
df_prediction.feature_names = df_prediction.columns.tolist()

# Access the custom attribute
print("Feature names attribute:", df_prediction.feature_names)
```

```
Feature names attribute: ['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping', 'Actual Rating', 'num_pages', 'ratings_count', ' biography', ' children', ' comics', ' cri
me', ' fantasy', ' fiction', ' graphic', ' historical fiction', ' history', ' mystery', ' nonfiction', ' paranormal', ' poetry', ' romance', ' thriller', ' youngadult', 'childre
n', 'comics', 'fantasy', 'fiction', 'mystery', 'nonfiction', 'poetry', 'romance', 'youngadult']
```

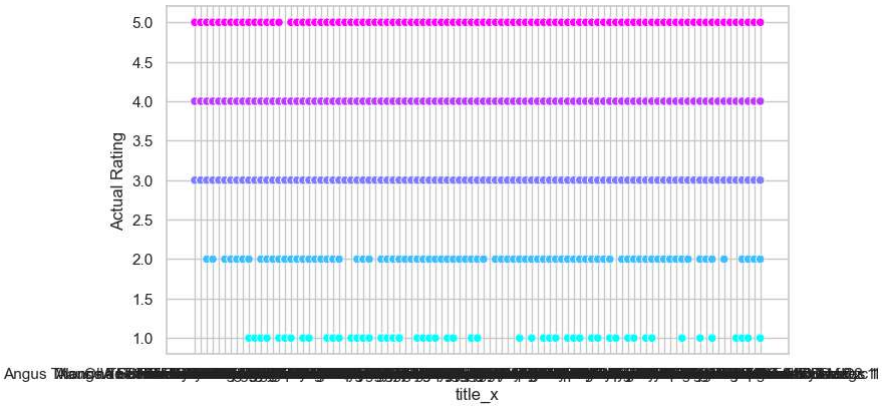
In [131…
```
print(df_prediction.head())
```

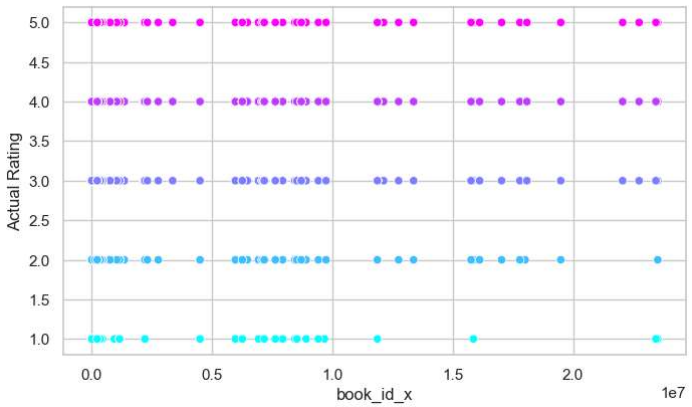
```
                      title_x  book_id_x  user_id_mapping  book_id_mapping  \
1   We Should All Be Feminists   22738563            45548              873
4   We Should All Be Feminists   22738563            41888              873
10  We Should All Be Feminists   22738563            19426              873
13  We Should All Be Feminists   22738563            46744              873
14  We Should All Be Feminists   22738563            26977              873

    Actual Rating  num_pages  ratings_count   biography   children   comics  \
1               4       1007         151473           0          0        0
4               5       1007         151473           0          0        0
10              5       1007         151473           0          0        0
13              5       1007         151473           0          0        0
14              4       1007         151473           0          0        0

    ...   youngadult  children  comics  fantasy  fiction  mystery  nonfiction  \
1   ...            0         0       0        1        0        0           0
4   ...            0         0       0        1        0        0           0
10  ...            0         0       0        1        0        0           0
13  ...            0         0       0        1        0        0           0
14  ...            0         0       0        1        0        0           0

    poetry  romance  youngadult
1        0        0           0
4        0        0           0
10       0        0           0
13       0        0           0
14       0        0           0

[5 rows x 32 columns]
```

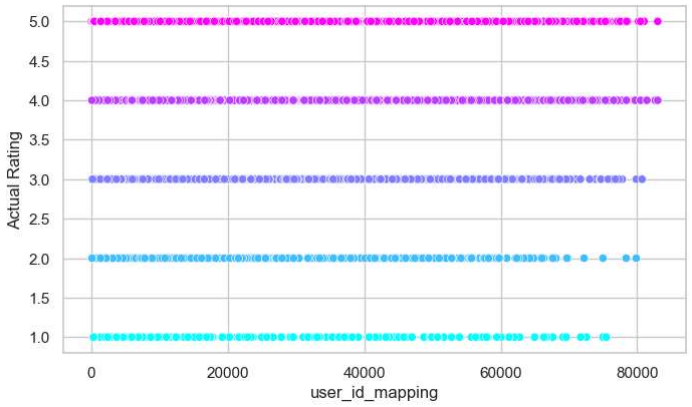
Plot the ratings (dependable variable) against each featuers (independent variable) to visualize potential patterns. Based on those plots, no patterns are visible.

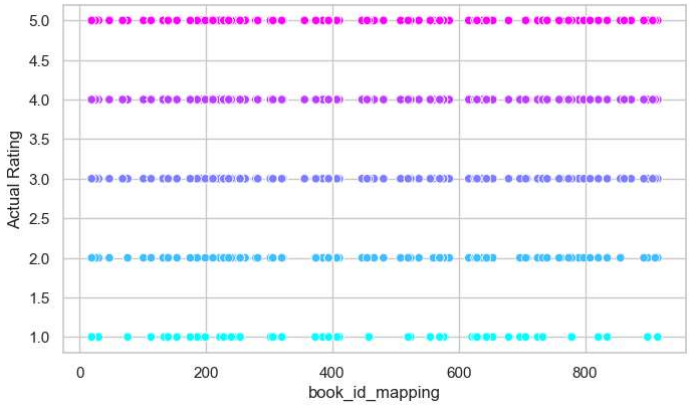
Therefore it was decided to reduce the data to 1 vector per user again, to use this as input for prediction.

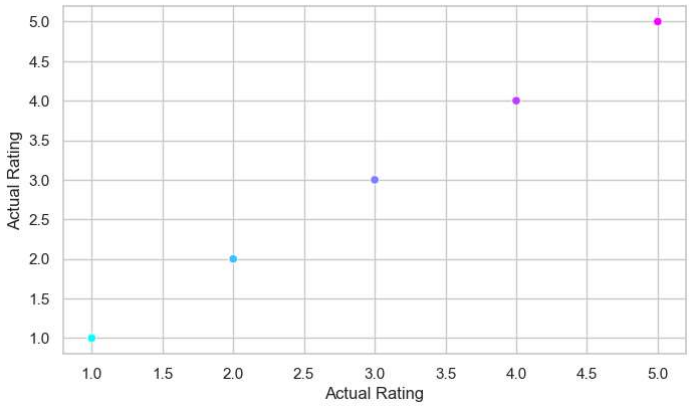
In [132…
```
# plotting ratings against each feature
for feature in df_prediction.feature_names:
    plt.figure(figsize=(8, 4.5))  # 8"-by-4.5" Figure
    sns.scatterplot(data=df_prediction, x=feature,
                    y='Actual Rating', hue='Actual Rating',
                    palette='cool', legend=False)
```

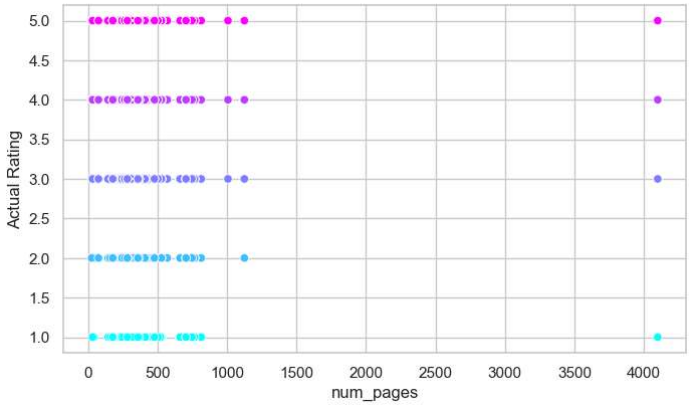


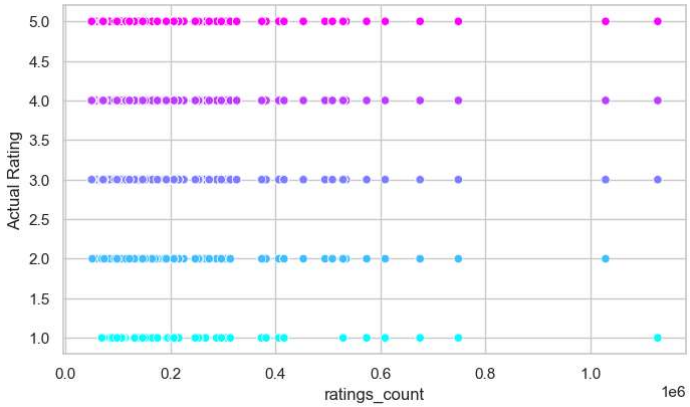


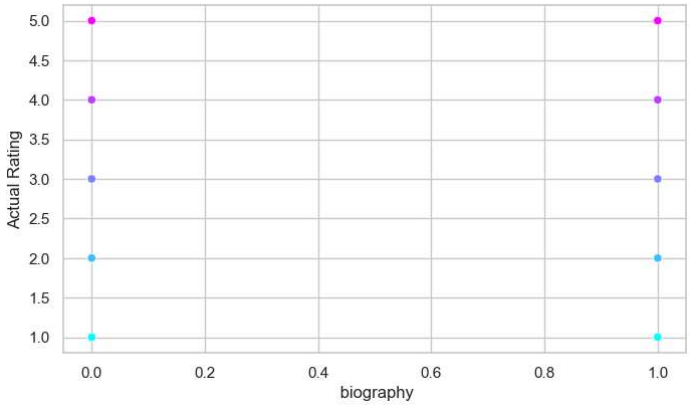


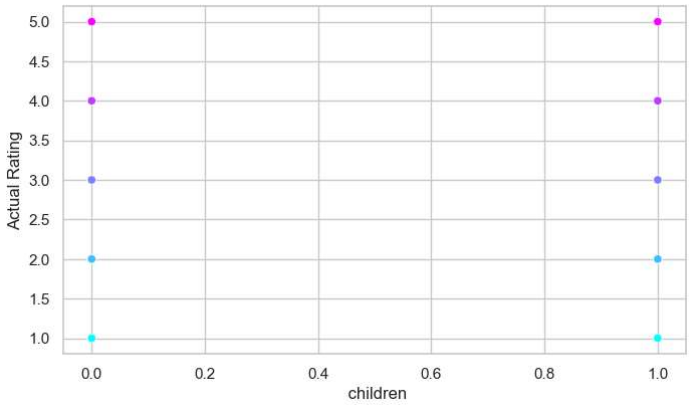

Actual Rating
book_id_mapping

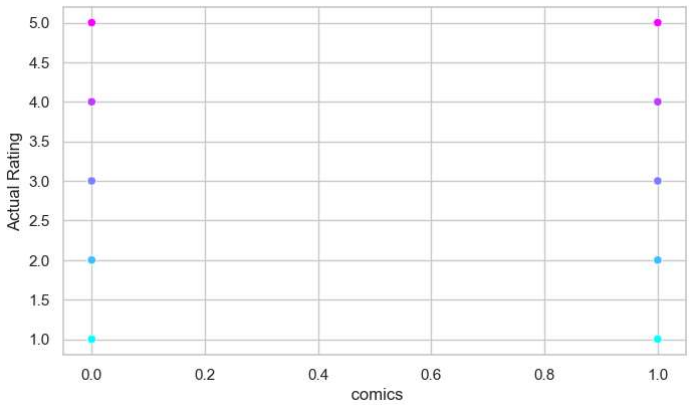

Actual Rating
Actual Rating

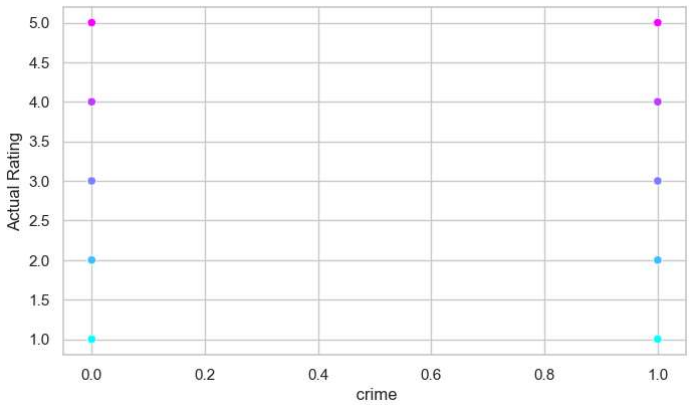

Actual Rating
num_pages

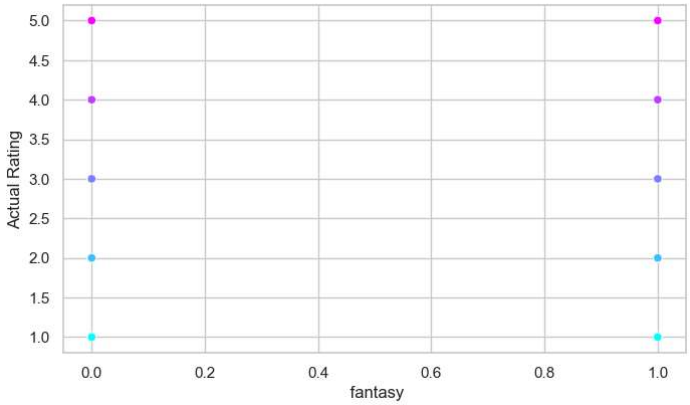

Actual Rating
ratings_count
1e6

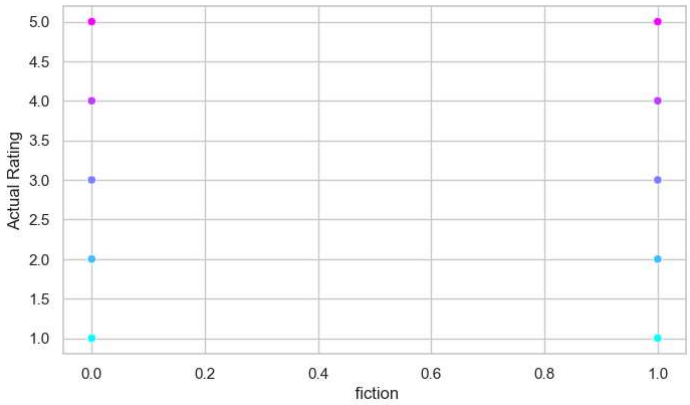

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
biography

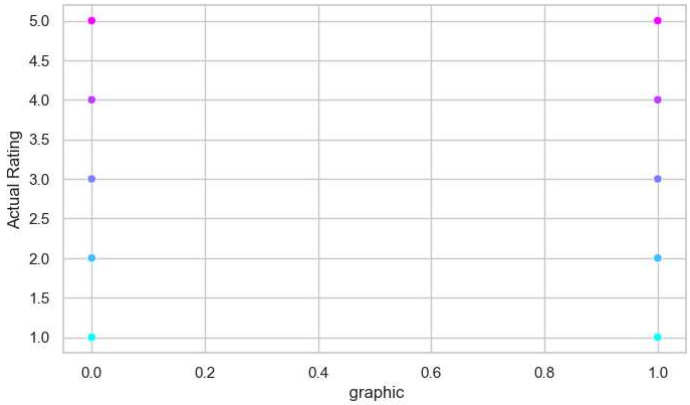

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
children

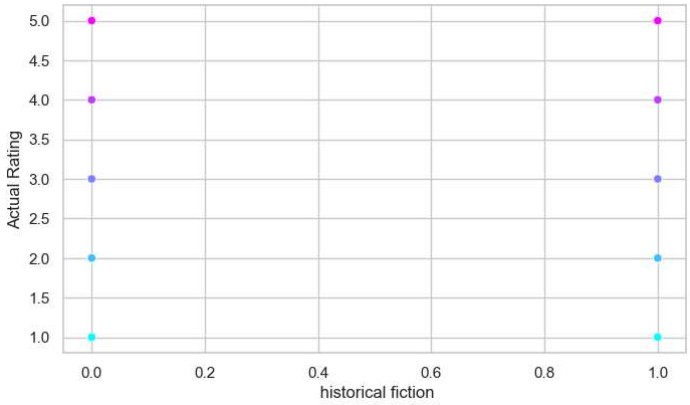

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
comics

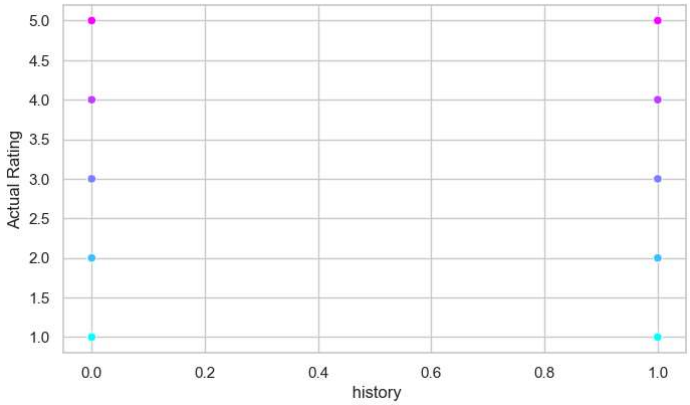

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
crime

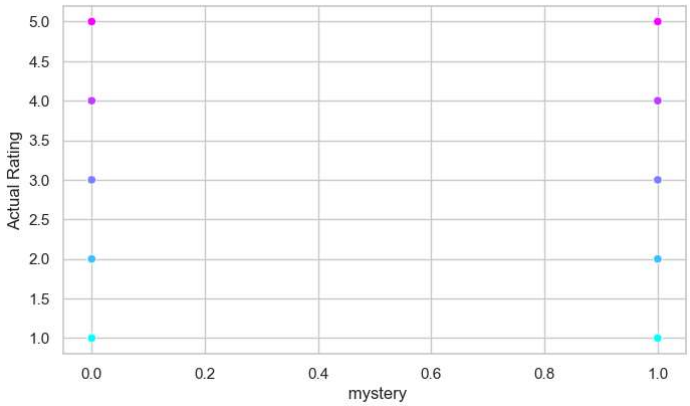

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
fantasy

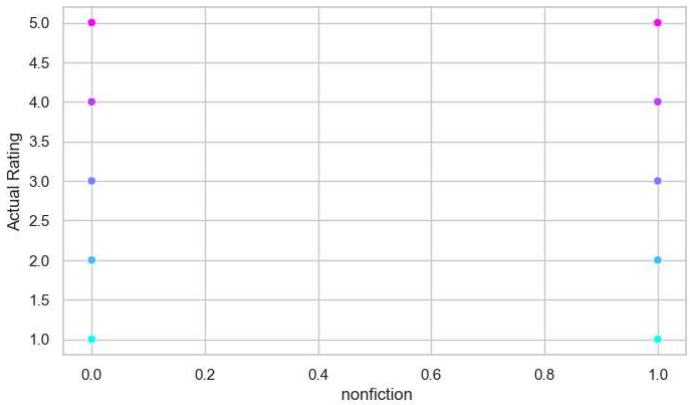

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
fiction

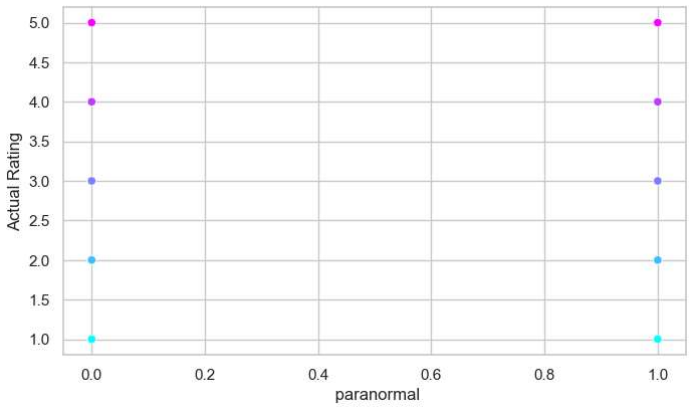

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
graphic

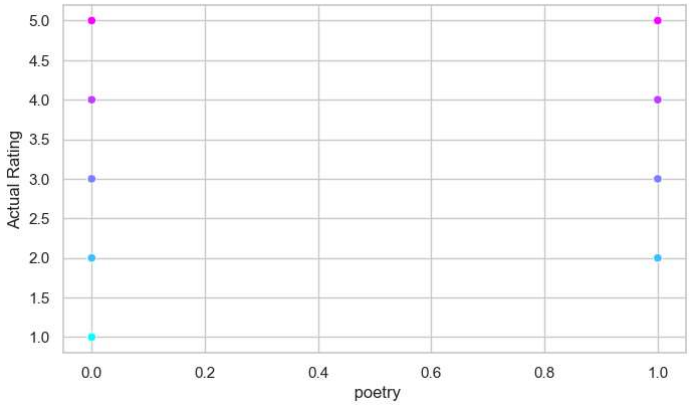

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
historical fiction

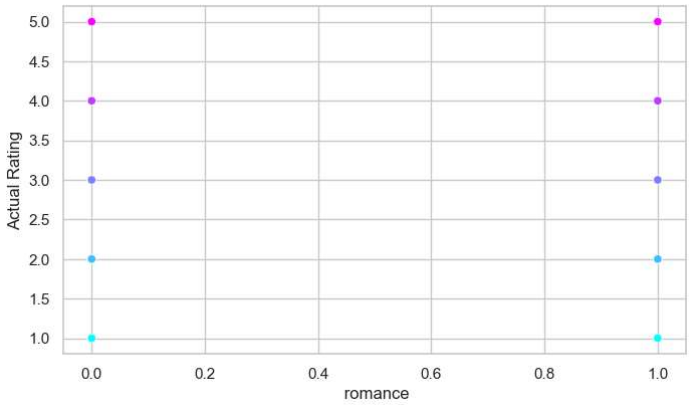

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
history

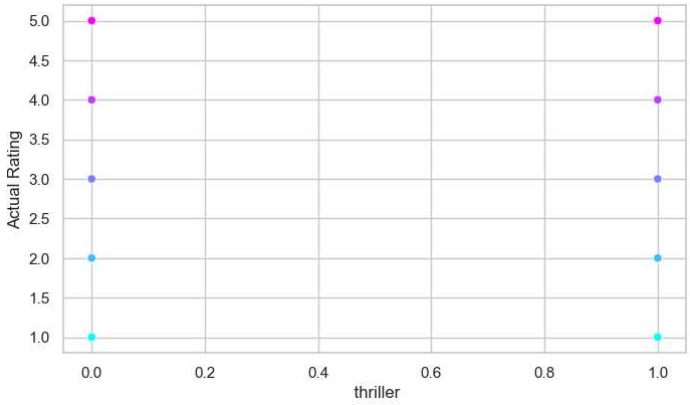

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
mystery

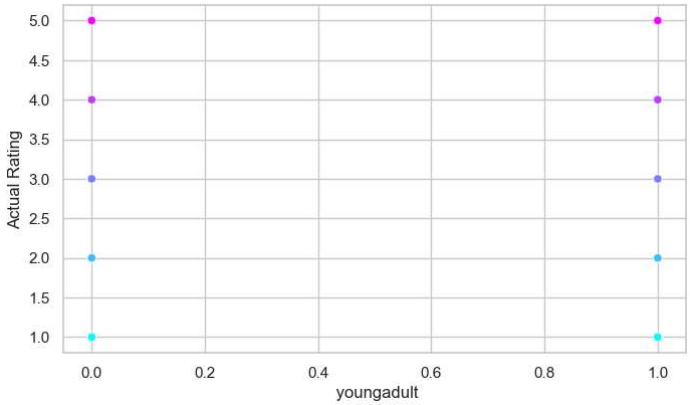

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
nonfiction

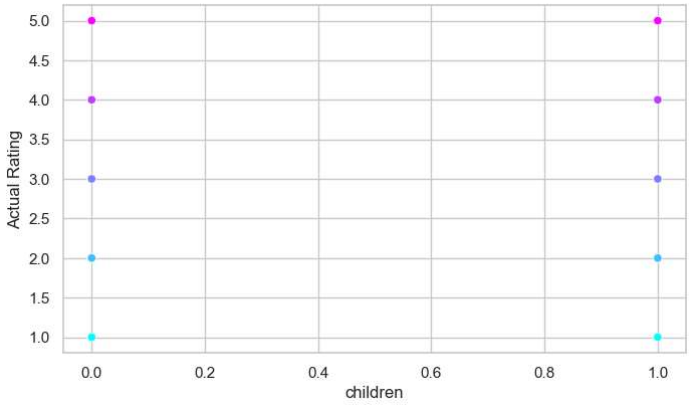

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
paranormal

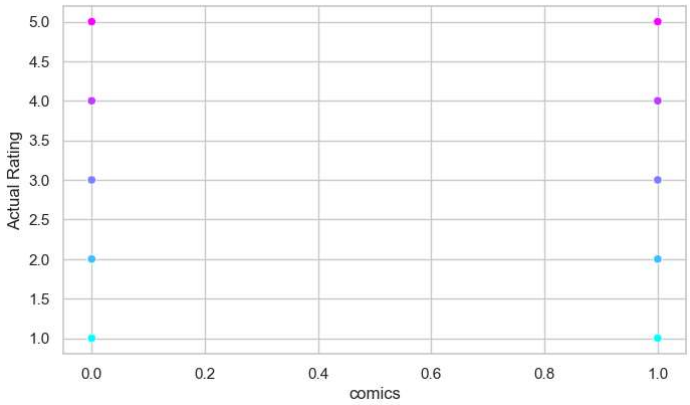

Actual Rating
poetry

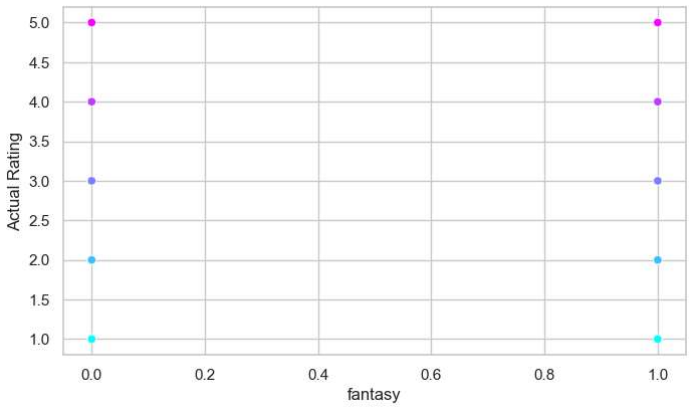

Actual Rating
romance

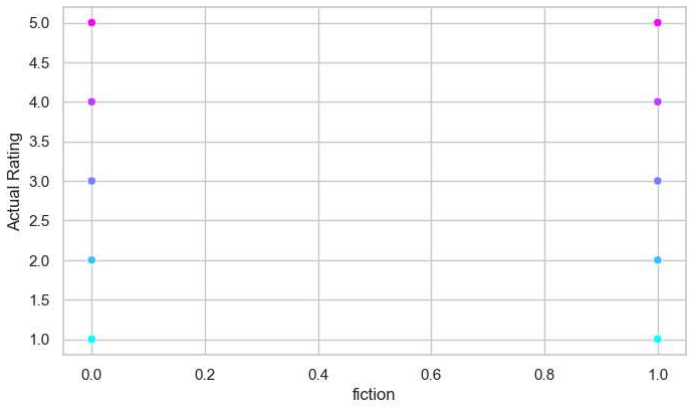

Actual Rating
thriller

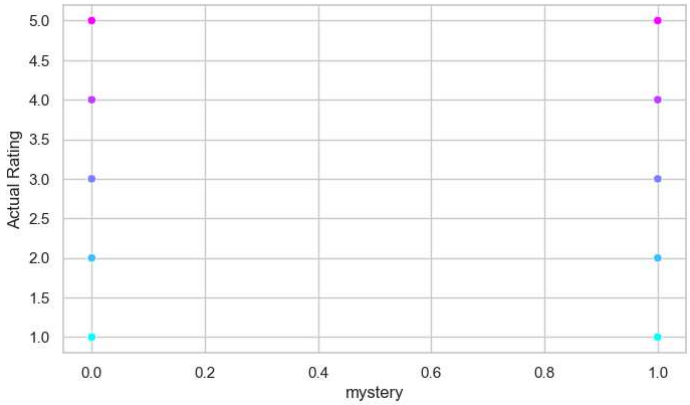

Actual Rating
youngadult

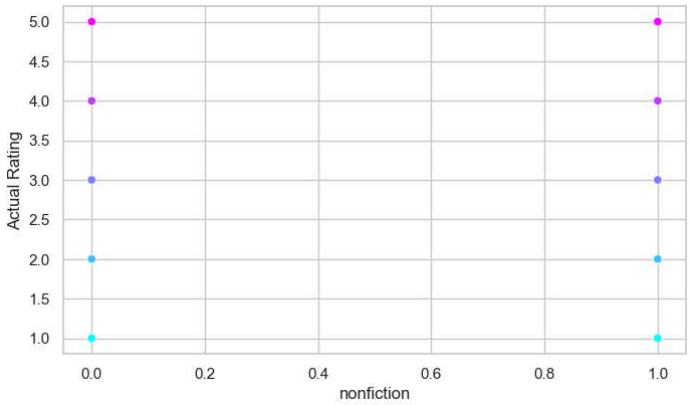

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
children

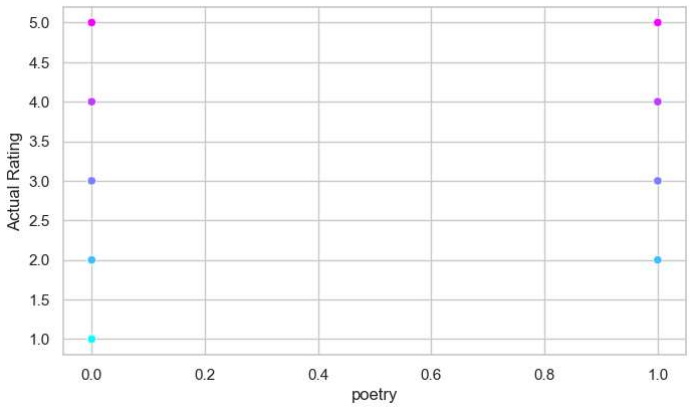

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
comics

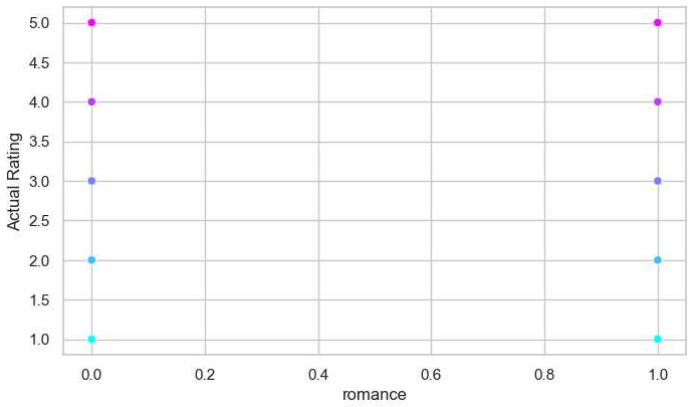

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
fantasy

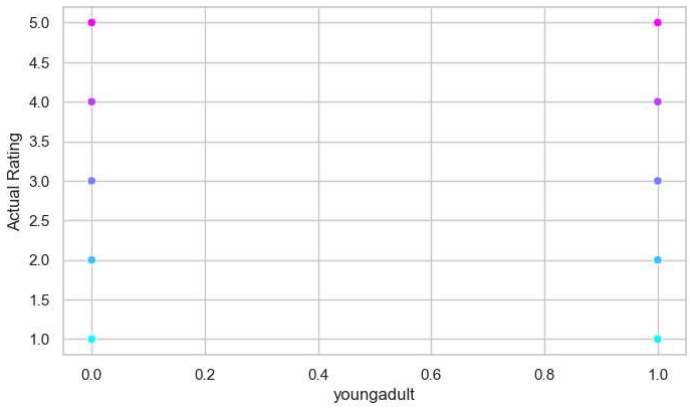

Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
fiction


Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
mystery


Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
nonfiction


Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
poetry


Actual Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
romance



In [133…
```
print(df_filtered.shape)
print(df_filtered.columns)
```

```
(5156, 39)
Index(['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping',
       'Predicted Rating', 'Actual Rating', 'book_id_y', 'title_y',
       'image_url', 'url', 'num_pages', 'ratings_count', 'description', 'name',
       ' biography', ' children', ' comics', ' crime', ' fantasy', ' fiction',
       ' graphic', ' historical fiction', ' history', ' mystery',
       ' nonfiction', ' paranormal', ' poetry', ' romance', ' thriller',
       ' youngadult', 'children', 'comics', 'fantasy', 'fiction', 'mystery',
       'nonfiction', 'poetry', 'romance', 'youngadult'],
      dtype='object')
```

In [134…
```
#remove columns that are not needed
df_for_prediction = df_filtered.drop(['Predicted Rating', 'book_id_y', 'title_y', 'description', 'image_url', 'url', 'name'],axis=1 )
```

In [135…
```
#verify that the columns are removed
print(df_for_prediction.shape)
print(df_for_prediction.columns)
```

```
(5156, 32)
Index(['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping',
       'Actual Rating', 'num_pages', 'ratings_count', ' biography',
       ' children', ' comics', ' crime', ' fantasy', ' fiction', ' graphic',
       ' historical fiction', ' history', ' mystery', ' nonfiction',
       ' paranormal', ' poetry', ' romance', ' thriller', ' youngadult',
       'children', 'comics', 'fantasy', 'fiction', 'mystery', 'nonfiction',
       'poetry', 'romance', 'youngadult'],
      dtype='object')
```

In [136…
```
user_metadata_df = df_filtered.copy()
```

In [137…
```
print(user_metadata_df.shape)
```

```
(5156, 39)
```

In [138…
```
# make sure all columns are numeric columns
df_for_prediction = df_for_prediction.apply(pd.to_numeric, errors='coerce')
```

In [139…
```
print(df_for_prediction.shape)
```

```
(5156, 32)
```

Creating user vectors based on different methods

- mean
- median
- sum

In [140…
```
#user vectors using mean
user_vectors_mean = df_for_prediction.groupby('user_id_mapping').mean()
print(user_vectors_mean.head())
```

```
                 title_x  book_id_x  book_id_mapping  Actual Rating  \
user_id_mapping
55                   NaN   315189.0            552.0            3.5
91                   NaN   195883.5            453.0            3.5
103                  NaN   596363.0            259.5            4.5
114                  NaN  3554811.5            202.0            2.5
117                  NaN     9296.0            403.0            2.5

                 num_pages  ratings_count   biography   children   comics  \
user_id_mapping
55                   179.5       168733.5         0.0        0.5      0.0
91                   183.0       157157.5         0.5        0.0      0.0
103                  738.0       341814.0         0.0        0.0      0.0
114                  368.0       289171.5         0.0        0.0      0.0
117                  339.5       161257.5         0.5        0.0      0.0

                  crime  ...   youngadult  children  comics  fantasy  fiction  \
user_id_mapping          ...
55                  0.0  ...          0.0       0.0     0.0      0.0      0.0
91                  0.5  ...          0.5       0.0     0.0      0.5      0.5
103                 0.0  ...          1.0       0.0     0.0      1.0      0.0
114                 0.5  ...          1.0       0.0     0.0      1.0      0.0
117                 0.5  ...          0.5       0.0     0.0      0.0      0.0

                 mystery  nonfiction  poetry  romance  youngadult
user_id_mapping
55                   0.0         0.0     0.0      0.0         1.0
91                   0.0         0.0     0.0      0.0         0.0
103                  0.0         0.0     0.0      0.0         0.0
114                  0.0         0.0     0.0      0.0         0.0
117                  0.5         0.0     0.0      0.5         0.0

[5 rows x 31 columns]
```

In [141…
```
#user vector using sum
user_vectors_sum = df_for_prediction.groupby('user_id_mapping').sum()
print(user_vectors_sum.head())
```

```
                title_x  book_id_x  book_id_mapping  Actual Rating  \
user_id_mapping
55                  0.0     630378             1104              7
91                  0.0     391767              906              7
103                 0.0    1192726              519              9
114                 0.0    7109623              404              5
117                 0.0      18592              806              5

                num_pages  ratings_count   biography   children   comics  \
user_id_mapping
55                    359         337467           0          1        0
91                    366         314315           1          0        0
103                  1476         683628           0          0        0
114                   736         578343           0          0        0
117                   679         322515           1          0        0

                 crime  ...   youngadult  children  comics  fantasy  fiction  \
user_id_mapping         ...
55                   0  ...            0         0       0        0        0
91                   1  ...            1         0       0        1        1
103                  0  ...            2         0       0        2        0
114                  1  ...            2         0       0        2        0
117                  1  ...            1         0       0        0        0

                mystery  nonfiction  poetry  romance  youngadult
user_id_mapping
55                    0           0       0        0           2
91                    0           0       0        0           0
103                   0           0       0        0           0
114                   0           0       0        0           0
117                   1           0       0        1           0

[5 rows x 31 columns]
```

In [142…

```
# user vectors using median

user_vectors_median = df_for_prediction.groupby('user_id_mapping').median()
print(user_vectors_median.head())
```

```
                title_x  book_id_x  book_id_mapping  Actual Rating  \
user_id_mapping
55                  NaN   315189.0            552.0            3.5
91                  NaN   195883.5            453.0            3.5
103                 NaN   596363.0            259.5            4.5
114                 NaN  3554811.5            202.0            2.5
117                 NaN     9296.0            403.0            2.5

                num_pages  ratings_count   biography   children   comics  \
user_id_mapping
55                  179.5       168733.5         0.0        0.5      0.0
91                  183.0       157157.5         0.5        0.0      0.0
103                 738.0       341814.0         0.0        0.0      0.0
114                 368.0       289171.5         0.0        0.0      0.0
117                 339.5       161257.5         0.5        0.0      0.0

                 crime  ...   youngadult  children  comics  fantasy  fiction  \
user_id_mapping         ...
55                 0.0  ...          0.0       0.0     0.0      0.0      0.0
91                 0.5  ...          0.5       0.0     0.0      0.5      0.5
103                0.0  ...          1.0       0.0     0.0      1.0      0.0
114                0.5  ...          1.0       0.0     0.0      1.0      0.0
117                0.5  ...          0.5       0.0     0.0      0.0      0.0

                mystery  nonfiction  poetry  romance  youngadult
user_id_mapping
55                  0.0         0.0     0.0      0.0         1.0
91                  0.0         0.0     0.0      0.0         0.0
103                 0.0         0.0     0.0      0.0         0.0
114                 0.0         0.0     0.0      0.0         0.0
117                 0.5         0.0     0.0      0.5         0.0

[5 rows x 31 columns]
```

Start the prediction

Get started with linear regression. no normalization. Do the same approach and use the same metrics for all 3 vectors to be able to judge, which vector performs best with this simple approach. Use this vector to tweak the approach and find out if the model anc be improved

In [143…

```
#the code I wanted to run did not go through due to missing values... finding the missing values now and cLeaning up
#check the missing values per column - title_x has missing values - therefore drop that column
missing_values = user_vectors_mean.isnull().sum()
print(missing_values)
```

```
title_x               2435
book_id_x                0
book_id_mapping          0
Actual Rating            0
num_pages                0
ratings_count            0
 biography               0
 children                0
 comics                  0
 crime                   0
 fantasy                 0
 fiction                 0
 graphic                 0
 historical fiction      0
 history                 0
 mystery                 0
 nonfiction              0
 paranormal              0
 poetry                  0
 romance                 0
 thriller                0
 youngadult              0
children                 0
comics                   0
fantasy                  0
fiction                  0
mystery                  0
nonfiction               0
poetry                   0
romance                  0
youngadult               0
dtype: int64
```

In [144…

```
#column title_x has missing values, so I drop it
user_vectors_mean = user_vectors_mean.drop(columns=['title_x'])
```

In [145…
```
print(user_vectors_mean.shape)
```

```
(2435, 30)
```

Running the first prediction (see initial setting above) for user_vector_mean

Outcome: not satisfying

Mean Squared Error: 0.6478485122777936

R-squared: 0.0412460789151855

In [146…
```
# Separate features (X) and target (y)

X = user_vectors_mean.drop(columns=['Actual Rating'])
y = user_vectors_mean['Actual Rating']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

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

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.6478485122777937
R-squared: 0.04124607891518528
```

Running the first prediction (see initial setting above) for user_vector_sum

Result is better than with mean, but not yet satisfying

Mean Squared Error: 2.8747540695971443

R-squared: 0.37543279109204053

In [147…
```
#check the missing values per column - title_x has missing values - therefore drop that column
missing_values = user_vectors_sum.isnull().sum()
print(missing_values)
```

```
title_x               0
book_id_x             0
book_id_mapping       0
Actual Rating         0
num_pages             0
ratings_count         0
 biography            0
 children             0
 comics               0
 crime                0
 fantasy              0
 fiction              0
 graphic              0
 historical fiction   0
 history              0
 mystery              0
 nonfiction           0
 paranormal           0
 poetry               0
 romance              0
 thriller             0
 youngadult           0
children              0
comics                0
fantasy               0
fiction               0
mystery               0
nonfiction            0
poetry                0
romance               0
youngadult            0
dtype: int64
```

In [148…
```
# Separate features (X) and target (y)

X = user_vectors_sum.drop(columns=['Actual Rating'])
y = user_vectors_sum['Actual Rating']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

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

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 2.874754069597145
R-squared: 0.3754327910920403
```

Running the first prediction (see initial setting above) for user_vector_median

Result is better than with mean, but not yet satisfying either

Mean Squared Error: 0.6683919894292838 R-squared: 0.03712562568513289

In [149…
```
#check the missing values per column - title_x has missing values - therefore drop that column
missing_values = user_vectors_median.isnull().sum()
print(missing_values)
```

```
title_x               2435
book_id_x                0
book_id_mapping          0
Actual Rating            0
num_pages                0
ratings_count            0
 biography               0
 children                0
 comics                  0
 crime                   0
 fantasy                 0
 fiction                 0
 graphic                 0
 historical fiction      0
 history                 0
 mystery                 0
 nonfiction              0
 paranormal              0
 poetry                  0
 romance                 0
 thriller                0
 youngadult              0
children                 0
comics                   0
fantasy                  0
fiction                  0
mystery                  0
nonfiction               0
poetry                   0
romance                  0
youngadult               0
dtype: int64
```

In [150…
```
#column title_x has missing values, so I drop it
user_vectors_median = user_vectors_median.drop(columns=['title_x'])
```

In [151…
```
# Separate features (X) and target (y)

X = user_vectors_median.drop(columns=['Actual Rating'])
y = user_vectors_median['Actual Rating']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

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

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.668391989429284
R-squared: 0.03712562568513256
```

Best model so far was with user_vectors_sum, therefore this model will be used going forward to improve the model

1st Iteration of trying to improve: normalizing/ standardizing the features

This is where we started: Mean Squared Error: 2.8747540695971443 R-squared: 0.37543279109204053

This is the result - it did not get better but rather worse Mean Squared Error: 2.8793309884843996 R-squared: 0.3744384126563185

In [152…
```
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit the model with scaled data
model.fit(X_train_scaled, y_train)

# Predict on the scaled test set
y_pred = model.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.6650363203622016
R-squared: 0.04195974668669278
```

Next iteration - apply ridge regression model with regularization

This is where we started: Mean Squared Error: 2.8747540695971443

R-squared: 0.37543279109204053

This is the result with tunining parameter alpha = 1.0 - it did get minimally better, not really worth the effort Mean Squared Error: 2.8737646386033306 R-squared: 0.3756477542294959

Try outs with 0.1 < alpha <10 leads to the results that R-squared does not get better than 0.375

In [153…
```
# Create a Ridge Regression model with regularization
ridge_model = Ridge(alpha=10)  # tune alpha

# Fit the model
ridge_model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = ridge_model.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.6681187063015077
R-squared: 0.03751931281009813
```

Next iteration with Lasso method

This is where we started: Mean Squared Error: 2.8747540695971443

R-squared: 0.37543279109204053

This is the result with tunining parameter alpha = 1.0 - it did get worse Mean Squared Error: 3.3486685307714454 R-squared: 0.2724704419272538

Try outs with minimizing alpha lead to a result nearing 0.375 again, so also not worth the effort

In [154…

```
# Create a Lasso Regression model
lasso_model = Lasso(alpha=0.0005)  # Tune alpha

# Fit and predict
lasso_model.fit(X_train_scaled, y_train)
y_pred = lasso_model.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.6676858598845863
R-squared: 0.03814286415341028
```

Next iteration Cross validation Before starting, the code again for user_vectors_sum and then for the scaler was executed again, to ensure the right data is used for the cross validation

This is where we started: Mean Squared Error: 2.8747540695971443 R-squared: 0.37543279109204053

This is the result, with a 5fold cross validation Cross-validated R-squared scores: [0.4414783 0.47124763 0.43147318 0.44211813 0.30602363] Mean R-squared: 0.4184681729038432

This is the result, with a 10fold cross validation Cross-validated R-squared scores: [0.31469016 0.45364744 0.47981629 0.43729431 0.4728822 0.39495923 0.36394261 0.49824312 0.32484085 0.41584173] Mean R-squared: 0.41561579448026603

Increasing the folds does not improve R-squared

Score with 3-fold cross validation Cross-validated R-squared scores: [0.46044342 0.37694054 0.422745] Mean R-squared: 0.42004298618043784 - this is so far the best result, but unfortunately not yet satisfying

In [155…

```
# Perform 5-fold cross-validation with R-squared as the scoring metric
cv_r2_scores = cross_val_score(model, X_train_scaled, y_train, cv=3, scoring='r2')

# Output the individual R-squared values for each fold
print(f'Cross-validated R-squared scores: {cv_r2_scores}')

# Output the mean R-squared value across all folds
print(f'Mean R-squared: {cv_r2_scores.mean()}')
```

```
Cross-validated R-squared scores: [ 0.02413477 -0.00505598  0.02043979]
Mean R-squared: 0.013172859386741917
```

Next iteration to improve the model - Feature selection

Evaluate if all the features are actually needed - with recursive feature elimination start with 10 features to select

Unfortunately, this approach is not improving the model

Outcome: Mean Squared Error: 4.21

R-squared: 0.09

In [156…

```
# Use RFE to select the best features
selector = RFE(model, n_features_to_select=10)
selector = selector.fit(X_train_scaled, y_train)

# Transform the datasets to only use selected features
X_train_rfe = selector.transform(X_train_scaled)
X_test_rfe = selector.transform(X_test_scaled)

# Fit the model with selected features
model.fit(X_train_rfe, y_train)

# Predict on the selected test set
y_pred = model.predict(X_test_rfe)

# evaluation for the model after feature selection

rfe_mse = mean_squared_error(y_test, y_pred)
rfe_r2 = r2_score(y_test, y_pred)

print(f'RFE Model - MSE: {rfe_mse:.2f}, R-squared: {rfe_r2:.2f}')
```

```
RFE Model - MSE: 0.67, R-squared: 0.04
```

Next iteration - Random Forest

with n_estimators=100, random_state=42 the results are not satisfying

Different variations of the parameters were used and tried, without satisfying results.

Mean Squared Error: 3.388927720124742

R-squared: 0.26372375650006896

In [157…

```
# Create a Random Forest model
rf_model = RandomForestRegressor(n_estimators=20, random_state=42)

# Fit the model
rf_model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = rf_model.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score
```

```
# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.7437776069504088
R-squared: -0.07147364009148482
```

Final iteration - polynomial features

with polynomial degree = 2 the following results are received

MSE: 3.37

R-squared: 0.27

Experimenting with higher degrees is not leading to better results.

In [158…

```
# Add polynomial features
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

# Fit the model with polynomial features
model.fit(X_train_poly, y_train)

# Predict on the test set
y_pred = model.predict(X_test_poly)

# Evaluate model performance
# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)


# R-squared (R²)
r2 = r2_score(y_test, y_pred)

# Print the results
print(f'MSE: {mse:.2f}')

print(f'R-squared: {r2:.2f}')
```

```
MSE: 1481362667437593041305 60.00
R-squared: -213402640082550707453952.00
```

K Nearest Neighbor Prediction

start with user_vectors_sum again, as it had the best starting values

This is where we start Mean Squared Error: 2.8747540695971443 R-squared: 0.37543279109204053

Score with 3-fold cross validation Cross-validated R-squared scores: [0.46044342 0.37694054 0.422745] Mean R-squared: 0.42004298618043784 - this is so far the best result, but unfortunately not yet satisfying

Results with test_size=0.2, random_state=42 and k=5 Mean Squared Error: 3.94 Root Mean Squared Error: 1.99 R-squared: 0.14

Changing k does not lead to better results

Trying out different distances 1st try - Minkowski distance - does not lead to any changes 2nd try - Manhatten distance - that is increasing R-squared to 0.15 with k=15, but also not helpful

In [159…

```
# Separate features and target
X = user_vectors_sum.drop(columns=['Actual Rating'])
y = user_vectors_sum['Actual Rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train the KNN regressor
k = 15

# Using Minkowski distance and using different p
#knn_minkowski = KNeighborsRegressor(n_neighbors=k, metric='minkowski', p=3)
#knn_minkowski.fit(X_train_scaled, y_train)
#y_pred_minkowski = knn_minkowski.predict(X_test_scaled)

#knn = KNeighborsRegressor(n_neighbors=k)
#knn.fit(X_train_scaled, y_train)

# Make predictions on the test set
#y_pred = knn.predict(X_test_scaled)

# Using Manhattan distance
knn_manhattan = KNeighborsRegressor(n_neighbors=k, metric='manhattan')
knn_manhattan.fit(X_train_scaled, y_train)
y_pred_manhattan = knn_manhattan.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred_manhattan)
rmse = mean_squared_error(y_test, y_pred_manhattan, squared=False)  # RMSE
r2 = r2_score(y_test, y_pred_manhattan)

# Print the results
print(f'Mean Squared Error: {mse:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'R-squared: {r2:.2f}')
```

```
Mean Squared Error: 3.90
Root Mean Squared Error: 1.97
R-squared: 0.15
```

Multi Layer Perceptron for prediction start with user_vectors_sum again, as it had the best starting values

This is where we start Mean Squared Error: 2.8747540695971443 R-squared: 0.37543279109204053

Score with 3-fold cross validation Cross-validated R-squared scores: [0.46044342 0.37694054 0.422745] Mean R-squared: 0.42004298618043784 - this is so far the best result, but unfortunately not yet satisfying

In [160…

```
# Separate features and target
X = user_vectors_sum.drop(columns=['Actual Rating'])
y = user_vectors_sum['Actual Rating']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train the MLP regressor

model = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', max_iter=500, random_state=42)
model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE
r2 = r2_score(y_test, y_pred)

# Print the results
print(f'MSE: {mse:.2f}')
print(f'RMSE: {rmse:.2f}')
print(f'R-squared: {r2:.2f}')
```

```
MSE: 4.45
RMSE: 2.11
R-squared: 0.03
```

Prediction for users with at least 3 reviews in the data

Hypothesis was, that if only users are used for prediction that have at least 3 book ratings, the data should be better and predictions should be better. As the execution shows, this is not true and the metrics are worse than with the dataset that also contains users with only 2 ratings.

In [161…

```
# group by the user IDs and count the entries per userid
id_counts = merged_df.groupby('user_id_mapping').size()
print(id_counts)
```

```
user_id_mapping
2        1
4        1
7        1
21       1
23       1
        ..
84173    1
84179    1
84212    1
84260    1
84282    1
Length: 17103, dtype: int64
```

The dataframe contains 254 users with 3 or more ratings, all in all 794 ratings remain.

In [162…

```
# filter for user-ids with 3 or more entries

ids_with_multiple_entries = id_counts[id_counts >= 3].index

# Filter the original DataFrame to only include these IDs
df_filtered3 = merged_df[merged_df['user_id_mapping'].isin(ids_with_multiple_entries)]
print(ids_with_multiple_entries.shape)
print(df_filtered3.shape)
```

```
(254,)
(794, 39)
```

Create 1 vector per user with the remaining users, re-using the code from before

In [163…

```
print(df_filtered3.shape)
print(df_filtered3.columns)
```

```
(794, 39)
Index(['title_x', 'book_id_x', 'user_id_mapping', 'book_id_mapping',
       'Predicted Rating', 'Actual Rating', 'book_id_y', 'title_y',
       'image_url', 'url', 'num_pages', 'ratings_count', 'description', 'name',
       ' biography', ' children', ' comics', ' crime', ' fantasy', ' fiction',
       ' graphic', ' historical fiction', ' history', ' mystery',
       ' nonfiction', ' paranormal', ' poetry', ' romance', ' thriller',
       ' youngadult', 'children', 'comics', 'fantasy', 'fiction', 'mystery',
       'nonfiction', 'poetry', 'romance', 'youngadult'],
      dtype='object')
```

In [164…

```
#need to drop some columns as they do not contain text
df_for_prediction3 = df_filtered3.drop(['Predicted Rating', 'book_id_y', 'title_y', 'title_x', 'description', 'image_url', 'url', 'name'],axis=1 )
print(df_for_prediction3.shape)
```

```
(794, 31)
```

In [165…

```
print(df_for_prediction3.dtypes)
```

```
book_id_x              int64
user_id_mapping        int64
book_id_mapping        int64
Actual Rating          int64
num_pages              int64
ratings_count          int64
 biography             int64
 children              int64
 comics                int64
 crime                 int64
 fantasy               int64
 fiction               int64
 graphic               int64
 historical fiction    int64
 history               int64
 mystery               int64
 nonfiction            int64
 paranormal            int64
 poetry                int64
 romance               int64
 thriller              int64
 youngadult            int64
children               int64
comics                 int64
fantasy                int64
fiction                int64
mystery                int64
nonfiction             int64
poetry                 int64
romance                int64
youngadult             int64
dtype: object
```

Following the same approach as before, 3 different user vectors are created, following three different approaches

In [166…
```
#user vectors using mean
user_vectors_mean3 = df_for_prediction3.groupby('user_id_mapping').mean()
print(user_vectors_mean3.head())
print(user_vectors_mean3.shape)
```

```
                  book_id_x  book_id_mapping  Actual Rating   num_pages  \
user_id_mapping
227            1.658567e+04       596.333333       3.333333  419.333333
451            4.210061e+06       248.666667       4.000000  476.666667
456            4.696680e+06       389.000000       4.000000  379.666667
532            3.951349e+06       489.250000       4.000000  312.250000
736            9.096000e+03       729.666667       4.000000  139.666667

                ratings_count   biography   children    comics     crime  \
user_id_mapping
227              143165.333333    0.000000   0.000000  0.000000  0.333333
451              399936.666667    0.333333   0.666667  0.000000  0.333333
456              458762.666667    0.333333   0.000000  0.000000  0.333333
532              224884.000000    0.000000   0.250000  0.000000  0.000000
736              158479.333333    0.000000   0.000000  0.333333  0.333333

                 fantasy  ...   youngadult  children    comics   fantasy  \
user_id_mapping           ...
227             0.333333  ...     0.000000  0.000000  0.000000  0.333333
451             0.000000  ...     0.666667  0.000000  0.000000  0.666667
456             0.333333  ...     0.000000  0.000000  0.000000  0.000000
532             0.000000  ...     1.000000  0.000000  0.000000  0.500000
736             0.666667  ...     0.666667  0.333333  0.333333  0.333333

                 fiction  mystery  nonfiction  poetry  romance  youngadult
user_id_mapping
227             0.333333      0.0    0.333333     0.0      0.0         0.0
451             0.000000      0.0    0.333333     0.0      0.0         0.0
456             0.666667      0.0    0.333333     0.0      0.0         0.0
532             0.000000      0.0    0.000000     0.0      0.5         0.0
736             0.000000      0.0    0.000000     0.0      0.0         0.0

[5 rows x 30 columns]
(254, 30)
```

In [167…
```
#user vectors using sum
user_vectors_sum3 = df_for_prediction3.groupby('user_id_mapping').sum()
print(user_vectors_sum3)
```

```
                book_id_x  book_id_mapping  Actual Rating  num_pages  \
user_id_mapping
227                49757             1789             10       1258
451             12630182              746             12       1430
456             14090040             1167             12       1139
532             15805395             1957             16       1249
736                27288             2189             12        419
...                  ...              ...            ...        ...
67611            2256616              740             13        817
69119           26442904              698             10       1264
71151           17854540              556             12       1323
75471              38133             2458             14        791
78622           19038110             1087             15       1753

                ratings_count   biography   children   comics   crime  \
user_id_mapping
227                    429496           0          0        0       1
451                   1199810           1          2        0       1
456                   1376288           1          0        0       1
532                    899536           0          1        0       0
736                    475438           0          0        1       1
...                       ...         ...        ...      ...     ...
67611                 1216767           1          1        1       1
69119                  674687           0          0        0       1
71151                  686306           0          0        0       2
75471                  617866           1          0        0       0
78622                  793115           1          1        0       1

                fantasy  ...   youngadult  children  comics  fantasy  \
user_id_mapping          ...
227                   1  ...            0         0       0        1
451                   0  ...            2         0       0        2
456                   1  ...            0         0       0        0
532                   0  ...            4         0       0        2
736                   2  ...            2         1       1        1
...                 ...  ...          ...       ...     ...      ...
67611                 0  ...            1         0       0        0
69119                 1  ...            2         0       0        1
71151                 1  ...            2         0       0        2
75471                 2  ...            4         2       1        1
78622                 1  ...            2         0       0        2

                fiction  mystery  nonfiction  poetry  romance  youngadult
user_id_mapping
227                   1        0           1       0        0           0
451                   0        0           1       0        0           0
456                   2        0           1       0        0           0
532                   0        0           0       0        2           0
736                   0        0           0       0        0           0
...                 ...      ...         ...     ...      ...         ...
67611                 1        0           1       0        0           1
69119                 0        0           0       0        1           1
71151                 0        0           0       0        0           1
75471                 0        0           0       0        0           0
78622                 0        0           0       0        0           1

[254 rows x 30 columns]
```

In [168…

```
#user vectors using median
user_vectors_median3 = df_for_prediction3.groupby('user_id_mapping').median()
print(user_vectors_median3.head())
```

```
                book_id_x  book_id_mapping  Actual Rating  num_pages  \
user_id_mapping
227              19691.0            621.0            3.0      528.0
451             452306.0            254.0            5.0      343.0
456            2213661.0            306.0            4.0      320.0
532            4067921.5            477.0            4.0      300.0
736               5096.0            735.0            5.0      144.0

                ratings_count   biography   children   comics   crime  \
user_id_mapping
227                   98088.0         0.0        0.0      0.0     0.0
451                  382518.0         0.0        1.0      0.0     0.0
456                  529274.0         0.0        0.0      0.0     0.0
532                  146117.5         0.0        0.0      0.0     0.0
736                  168436.0         0.0        0.0      0.0     0.0

                fantasy  ...   youngadult  children  comics  fantasy  \
user_id_mapping          ...
227                 0.0  ...          0.0       0.0     0.0      0.0
451                 0.0  ...          1.0       0.0     0.0      1.0
456                 0.0  ...          0.0       0.0     0.0      0.0
532                 0.0  ...          1.0       0.0     0.0      0.5
736                 1.0  ...          1.0       0.0     0.0      0.0

                fiction  mystery  nonfiction  poetry  romance  youngadult
user_id_mapping
227                 0.0      0.0         0.0     0.0      0.0         0.0
451                 0.0      0.0         0.0     0.0      0.0         0.0
456                 1.0      0.0         0.0     0.0      0.0         0.0
532                 0.0      0.0         0.0     0.0      0.5         0.0
736                 0.0      0.0         0.0     0.0      0.0         0.0

[5 rows x 30 columns]
```

Starting with prediction - linear regression for all 3 vectors

Interestingly, the results are worse than with the previous data set

In [169…

```
# Train the model for user_vector_mean
# Separate features (X) and target (y)

X = user_vectors_mean3.drop(columns=['Actual Rating'])
y = user_vectors_mean3['Actual Rating']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

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

# Evaluate the model performance
```

```
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.46241899789327806
R-squared: -0.10950699636737493
```

In [170…

```
# Train the model for user_vector_sum
# Separate features (X) and target (y)

X = user_vectors_sum3.drop(columns=['Actual Rating'])
y = user_vectors_sum3['Actual Rating']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

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

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 4.34629207703252
R-squared: 0.16348189341707986
```

In [171…

```
# Train the model for user_vector_median
# Separate features (X) and target (y)

X = user_vectors_median3.drop(columns=['Actual Rating'])
y = user_vectors_median3['Actual Rating']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

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

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.8114334562303864
R-squared: -0.2733263466999911
```

Continuing with the Vector with method sum as it had the best result, although not good at all

1st standardizing the data, then trying cross validation

Cross validation is not leading to promising results.

In [172…

```
#standardizing/ normalizing the features first, start with running the regression for user_vector_sum3 again
from sklearn.preprocessing import StandardScaler

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit the model with scaled data
model.fit(X_train_scaled, y_train)

# Predict on the scaled test set
y_pred = model.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.8114334562303879
R-squared: -0.27332634669999334
```

In [173…

```
# Perform 5-fold cross-validation with R-squared as the scoring metric
cv_r2_scores = cross_val_score(model, X_train_scaled, y_train, cv=2, scoring='r2')

# Output the individual R-squared values for each fold
print(f'Cross-validated R-squared scores: {cv_r2_scores}')

# Output the mean R-squared value across all folds
print(f'Mean R-squared: {cv_r2_scores.mean()}')
```

```
Cross-validated R-squared scores: [-1.24910210e+26 -5.47728277e-01]
Mean R-squared: -6.245510482636489 5e+25
```

Next iteration - apply Ridge Regression (run the scaler first again, just to be sure) start with alpha = 1

also not leading to better results than R-squared ca 0.4

In [174…

```
# Create a Ridge Regression model with regularization
ridge_model = Ridge(alpha=5)  # You can tune alpha (regularization strength)

# Fit the model
ridge_model.fit(X_train_scaled, y_train)
```

```
# Predict on the test set
y_pred = ridge_model.predict(X_test_scaled)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output the results
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# If you want to inspect the coefficients of the regression model
coefficients = model.coef_
print("Model coefficients:", coefficients)
```

```
Mean Squared Error: 0.7936668007071986
R-squared: -0.2454463641866811
Model coefficients: [ 1.55127639e-01  5.72400917e-02 -1.07667321e-01  1.14833278e-01
 -3.33175607e-02 -9.37484669e-02  1.57750131e-02  6.33070967e-02
  1.02579980e-01  7.83356422e-03 -2.46776492e-02 -3.33175607e-02
 -3.33175607e-02 -8.22144980e-02  1.38777878e-17 -8.87152332e-02
 -1.38777878e-17 -9.99143841e-02  6.33070967e-02  4.02086555e-02
 -5.61425976e-03  1.07865603e-01  1.70503940e-01 -7.83356422e-03
  0.00000000e+00  5.34941719e-02  0.00000000e+00 -1.40204182e-02
 -3.64212719e-02]
```

Next iteration - Feature selection

Best result so far with R-Squared= 0.26 when n=15 features, and MSE= 3.83, but this is also not good

In [175…

```
from sklearn.feature_selection import RFE

# Use RFE to select the best features
selector = RFE(model, n_features_to_select=15)
selector = selector.fit(X_train_scaled, y_train)

# Transform the datasets to only use selected features
X_train_rfe = selector.transform(X_train_scaled)
X_test_rfe = selector.transform(X_test_scaled)

# Fit the model with selected features
model.fit(X_train_rfe, y_train)

# Predict on the selected test set
y_pred = model.predict(X_test_rfe)

# evaluation for the model after feature selection

rfe_mse = mean_squared_error(y_test, y_pred)
rfe_r2 = r2_score(y_test, y_pred)

print(f'RFE Model - MSE: {rfe_mse:.2f}, R-squared: {rfe_r2:.2f}')
```

```
RFE Model - MSE: 0.81, R-squared: -0.27
```

Next iteration polynomials - unfortunately also no satisfying results (trying with 2-4 degrees)

In [176…

```
# Add polynomial features
poly = PolynomialFeatures(degree=4)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

# Fit the model with polynomial features
model.fit(X_train_poly, y_train)

# Predict on the test set
y_pred = model.predict(X_test_poly)

# Evaluate model performance
# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)


# R-squared (R²)
r2 = r2_score(y_test, y_pred)

# Print the results
print(f'MSE: {mse:.2f}')

print(f'R-squared: {r2:.2f}')
```

```
MSE: 72751808629932160.00
R-squared: -114164376619278160.00
```

Next iteration K-Nearest Neighbor

- with standard distance not leading to good results
- Manhattan distance also not leading to good results
- Minkowski distance also not leading to good results

In [177…

```
# Separate features and target
X = user_vectors_sum3.drop(columns=['Actual Rating'])
y = user_vectors_sum3['Actual Rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train the KNN regressor
k = 45

# Using Minkowski distance and using different p
knn_minkowski = KNeighborsRegressor(n_neighbors=k, metric='minkowski', p=3)
knn_minkowski.fit(X_train_scaled, y_train)
y_pred_minkowski = knn_minkowski.predict(X_test_scaled)

#knn = KNeighborsRegressor(n_neighbors=k)
#knn.fit(X_train_scaled, y_train)

# Make predictions on the test set
#y_pred = knn.predict(X_test_scaled)

# Using Manhattan distance
#knn_manhattan = KNeighborsRegressor(n_neighbors=k, metric='manhattan')
#knn_manhattan.fit(X_train_scaled, y_train)
#y_pred_manhattan = knn_manhattan.predict(X_test_scaled)
```

```
# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred_minkowski) #y_pred needs to be adjusted to which distance has been used in the previous code
rmse = mean_squared_error(y_test, y_pred_minkowski, squared=False)  #y_pred needs to be adjusted to which distance has been used in the previous code
r2 = r2_score(y_test, y_pred_minkowski) #y_pred needs to be adjusted to which distance has been used in the previous code

# Print the results
print(f'Mean Squared Error: {mse:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'R-squared: {r2:.2f}')
```

```
Mean Squared Error: 5.01
Root Mean Squared Error: 2.24
R-squared: 0.04
```

Insights and findings

(Heidi)

With all the experiments in the prediction based on user information how they rated books, from which we have a bit more information led to the best result of

Score with 3-fold cross validation Cross-validated R-squared scores: [0.46044342 0.37694054 0.422745]

Mean R-squared: 0.42004298618043784

While this is an interesting finding it is not yet satisfying.

It would be interesting to experiment with more data. Unfortunately the analysis was limited by the number of books for which metadata was available which was <100. An interesting next step could have been to get more metadata for more books and run the algorithms again and understand whether this would yield better results.

Another interesting angle could be to combine some of the methods and algorithms.

(Lea, copy from previous part)

- Comparing the trained models to the Predicted Rating included in the dataset, it is obvious that the model quality is lower than the one used by the dataset creator.
- Regression Trees seem to come the closest to the Predicted Rating (In previous runs (MSE: 0.9), not true anymore. Regression Neural Network is better now).
- SVR und SVM had major problems during training and weren't converging correctly. Scalers were suggested in the warnings and might have helped those issues. Time constraints did not allow for more fiddling with optimization.
- On average the prediction made by Regression models seems to have performed better. In the case of neural networks, the predictions seem of similar quality, so the path easiest to optimize and implement should be further optimized.
- The trained models include the user id of the reviewer as an input feature. This was determined to be unusable from the beginning, so confidence in the models was not high from the beginning. The user id was supposed to be exchanged for a representation of the users reading preferences created via clustering. As the clustering approaches proved non-conclusive in the time given, no replacement took place. Further analysis regarding representing the user by the books he enjoyed and didn't enjoy should take place. Averaging scores by genre or fuzzy clustering approaches could be evaluated.
- The reviews were unbalanced towards better scores, so boosting ensembles made from Regression models could prove more accurate
- It was determied that quite a bit more work could be put into preprocessing the data to observe the impact on the rating prediction
- A deep dive into the hyperparameters of models trained could prove helpful

Overall insights and findings

Throughout the work with the data we realized that the analysis and preparation of the data needed more time and effort than we initially assumed. We were able to adapt to that. Once the data was ready to be used in the models it was easy to apply, adjust and train different models, but their quality does not hold up to our standards.

The overall outcome is, that with all the experiments and different angles we used to analyse the data and create and train models to predict how a user would rate a book, we did not manage to find a model set up that yields acceptable results.

Unfortunately, this is an outcome that needs to be expected in data science projects.

Following potential next steps (isolated, as well as in combination) could be tried to still find and train a model that might lead to better results

- enrich the data with more book metadata
- combine some of the methods that were used individually
- try other methods
- create a conclusive user representation to be used in predicting ratings
- evaluate features used in prediction further
- optimize models for more fitting metric combinations instead of single metrics
- optimize pipelines instead of single models

But it would need to be important to first make a decsion how much time and effort should still be invested in such a project to not waste valuable resources.

Sources

- https://github.com/kapadias/medium-articles/blob/master/natural-language-processing/topic-modeling/Evaluate%20Topic%20Models.ipynb
- https://medium.com/@walter_sperat/using-optuna-with-sklearn-the-right-way-part-1-6b4ad0ab2451
- https://medium.com/@walter_sperat/using-optuna-with-sklearn-the-right-way-part-4-1e17eb2696fe
- https://learn-scikit.oneoffcoder.com/optuna.html
- https://learn-scikit.oneoffcoder.com/gensim.html
- https://stackoverflow.com/questions/60613532/how-do-i-calculate-the-coherence-score-of-an-sklearn-lda-model
- https://www.kaggle.com/code/muhammetgamal5/kfold-cross-validation-optuna-tuning
- https://github.com/optuna/optuna/issues/1862
- https://towardsdatascience.com/practical-guide-to-topic-modeling-with-lda-05cd6b027bdf
- https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/
- https://stackoverflow.com/questions/50785661/how-to-improve-performance-of-lda-latent-dirichlet-allocation-in-sci-kit-learn
- https://blog.mlreview.com/topic-modeling-with-scikit-learn-e80d33668730
- Deitel, Paul, and Harvey Deitel. *Intro to python for computer science and data science.* Pearson education, 2021.
- Lecture Scripts and Notebooks
- Documentation of Libraries used

Distribution of work and effort

Topic/ Task	Contributor(s)
Create project draft and plan	Both
Data Exploration (Code)	Lea Kleemann
Data Exploration (Interpretation)	Both
Clustering the users	Heidi Gehring

Topic/ Task	Contributor(s)
Clustering with DBScan	Both in pairing session
Topic Modeling	Lea Kleemann
Insights and findings for Clustering	Both
Prediction based on Topic Modeling	Lea Kleemann
Prediction based on users	Heidi Gehring
Final insights and findings	Both
Finalization of the assignment	Both