cleanup

May 13, 2024

[1]:
```python
import pandas as pd
import time
import datetime
from math import ceil
from os import path, makedirs
from sklearn.impute import KNNImputer
from sklearn.preprocessing import MinMaxScaler

pd.set_option('display.max_columns', None)
```

[2]:
```python
ratings_df = pd.read_csv('raw_data/rangering.dat', sep='::', header=0,
                         names=['BrukerID', 'FilmID', 'Rangering',␣
 ↪'Tidstempel'],
                         engine='python')

ratings_df.describe()
```

[2]:

	BrukerID	FilmID	Rangering	Tidstempel
count	900187.000000	900187.000000	900187.000000	8.986950e+05
mean	2991.864495	1989.675878	4.279477	9.722414e+08
std	1736.204837	1126.366532	1.971075	1.214672e+07
min	0.000000	0.000000	1.000000	9.567039e+08
25%	1458.000000	1037.000000	3.000000	9.653029e+08
50%	2967.000000	1959.000000	4.000000	9.729904e+08
75%	4501.000000	2963.000000	5.000000	9.752202e+08
max	6040.000000	3952.000000	10.000000	1.046455e+09

[3]:
```python
missing_vals = ratings_df.isnull().sum()
print(missing_vals, '\n')
perc = round(missing_vals / ratings_df.shape[0] * 100, 2)
print(f'There are {ratings_df.shape[0]} rows in the dataset.')
print(f'Proportion of missing data for each column in %: \n{perc}')
```

```
BrukerID         0
FilmID           0
Rangering        0
Tidstempel    1492
dtype: int64
```

```
There are 900187 rows in the dataset.
Proportion of missing data for each column in %:
BrukerID      0.00
FilmID        0.00
Rangering     0.00
Tidstempel    0.17
dtype: float64
```

[4]:
```
missing_df = ratings_df[ratings_df['Tidstempel'].isnull()]
not_missing_df = ratings_df[ratings_df['Tidstempel'].notnull()]

date = '01/08/2000 01:00:00'  # UTC +1
date_of_conversion = time.mktime(datetime.datetime.strptime(date, "%d/%m/%Y %H:
↪%M:%S").utctimetuple())
print('date_of_conversion', date_of_conversion)

old_scaling = not_missing_df[not_missing_df['Tidstempel'] < date_of_conversion]
new_scaling = not_missing_df[not_missing_df['Tidstempel'] >= date_of_conversion]

rated_before = old_scaling['BrukerID'].unique().tolist()
rated_after = new_scaling['BrukerID'].unique().tolist()
rated_before_and_after = []

for user in rated_before:
    if user in rated_after:
        rated_before_and_after.append(user)


rated_only_before = sorted(list(set(rated_before).
↪difference(rated_before_and_after)))
rated_only_after = sorted(list(set(rated_after).
↪difference(rated_before_and_after)))

imputed = 0
for row in missing_df.iterrows():
    index = row[0]
    user_id = int(row[1][0])

    if user_id in rated_only_before:
        avg_timestamp = old_scaling.loc[old_scaling['BrukerID'] == user_id,␣
↪'Tidstempel'].mean()
        ratings_df.loc[index, 'Tidstempel'] = avg_timestamp
        imputed += 1

    elif user_id in rated_only_after:
```

```
        avg_timestamp = new_scaling.loc[new_scaling['BrukerID'] == user_id,␣
↪'Tidstempel'].mean()
        ratings_df.loc[index, 'Tidstempel'] = avg_timestamp
        imputed += 1

print(f'Total number of missing values after imputing the avg timestamp for␣
↪{imputed} entries: ',
      ratings_df.isnull().sum().sum(), '\n')

missing_df = ratings_df[ratings_df['Tidstempel'].isnull()]
missing_and_high_rating = missing_df.loc[missing_df['Rangering'] > 5]
print('missing timestamp and high rating: ', len(missing_and_high_rating))

for row in missing_and_high_rating.iterrows():
    index = row[0]
    user_id = int(row[1][0])
    avg_timestamp = old_scaling.loc[:, 'Tidstempel'].mean()
    ratings_df.loc[index, 'Tidstempel'] = avg_timestamp

ratings_df.dropna(how='any', inplace=True)

print(f'Missing values after deleting the remaining entries with a rating < 6:␣
↪{ratings_df.isnull().sum().sum()}')


ratings_df['Tidstempel'] = ratings_df['Tidstempel'].astype(int)

old_scaling = ratings_df[ratings_df['Tidstempel'] < date_of_conversion]
new_scaling = ratings_df[ratings_df['Tidstempel'] >= date_of_conversion]

pd.reset_option('mode.chained_assignment')
with pd.option_context('mode.chained_assignment', None):

     Replacing the values from integers to strings, so that I can use regex on␣
↪them
    old_scaling['Rangering'].replace(to_replace=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                                     value=['1', '2', '3', '4', '5', '6', '7',␣
↪'8', '9', '10'], inplace=True)

    old_scaling['Rangering'].replace(regex=True, to_replace=['1\b|2', '3|4',␣
↪'5|6', '7|8', '9|10'],
                                     value=[1, 2, 3, 4, 5], inplace=True)


ratings_df_cleaned = old_scaling.append(new_scaling, ignore_index=True)
```

```
date_of_conversion 965088000.0
```

```
Total number of missing values after imputing the avg timestamp for 1359
entries:  133

missing timestamp and high rating:  63
Missing values after deleting the remaining entries with a rating < 6: 0
```

[5]:
```python
if path.exists('cleaned_data'):
    print('cleaned_data folder already exists')
else:
    makedirs('cleaned_data')

ratings_df_cleaned.to_csv('cleaned_data/rangering.csv', index=False)
```

```
cleaned_data folder already exists
```

[6]:
```python
users_df = pd.read_json('raw_data/bruker.json', orient='split')

users_df.head(5)
```

[6]:

	BrukerID	Kjonn	Alder	Jobb	Postkode
0	0	None	45.0	6.0	92103
1	1	M	50.0	16.0	55405-2546
2	2	M	18.0	20.0	44089
3	3	M	NaN	1.0	33304
4	4	M	35.0	6.0	48105

[7]:
```python
users_df.describe()
```

[7]:

	BrukerID	Alder	Jobb
count	6040.000000	5046.000000	5447.000000
mean	3020.465894	30.666072	9.104278
std	1743.799216	12.954723	11.239708
min	0.000000	1.000000	0.000000
25%	1510.750000	25.000000	3.000000
50%	3020.500000	25.000000	7.000000
75%	4530.250000	35.000000	14.000000
max	6040.000000	56.000000	99.000000

[8]:
```python
missing_vals = users_df.isnull().sum()
perc = round(missing_vals / users_df.shape[0] * 100, 2)
print(f'There are {users_df.shape[0]} rows in the dataset.')
print(f'Proportion of missing data for each column in %: \n{perc}')
```

```
There are 6040 rows in the dataset.
Proportion of missing data for each column in %:
BrukerID     0.00
Kjonn        5.02
Alder       16.46
```

```
Jobb          9.82
Postkode      7.47
dtype: float64
```

[9]:
```
def count_rows_with_n_missing_vals(dataframe, n):
    missing_val_count = dataframe.shape[0] - (dataframe.dropna(how='any',␣
↪thresh=(dataframe.shape[1] - n)+1).shape[0])
    return missing_val_count


print(f'There are {count_rows_with_n_missing_vals(users_df, 2)} rows with at␣
↪least 2 missing values')
print(f'There are {count_rows_with_n_missing_vals(users_df, 3)} rows with at␣
↪least 3 missing values')
```

```
There are 291 rows with at least 2 missing values
There are 20 rows with at least 3 missing values
```

[10]:
```
users_df['Jobb'].fillna(0, inplace=True)
users_df.head()
```

[10]:

	BrukerID	Kjonn	Alder	Jobb	Postkode
0	0	None	45.0	6.0	92103
1	1	M	50.0	16.0	55405-2546
2	2	M	18.0	20.0	44089
3	3	M	NaN	1.0	33304
4	4	M	35.0	6.0	48105

[11]:
```
users_df.replace(to_replace=['M', 'F'], value=[1, 0], inplace=True)

users_df['Postkode_5'] = users_df.Postkode.str[:3]

df = users_df.drop(['Postkode', 'BrukerID'], axis=1)

scaler = MinMaxScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

[12]:
```
imputer = KNNImputer(n_neighbors=5, )
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
print(df.isna().sum())
```

```
Kjonn         0
Alder         0
Jobb          0
Postkode_5    0
dtype: int64
```

[13]:
```
df = pd.DataFrame(scaler.inverse_transform(df), columns=df.columns)

df[['BrukerID', 'Postkode']] = users_df[['BrukerID', 'Postkode']]

users_df = df[['BrukerID', 'Kjonn', 'Alder', 'Jobb', 'Postkode', 'Postkode_5']]

for row in users_df.iterrows():
    index = row[0]
    gender = row[1][1]
    age = row[1][2]
    job = row[1][3]
    postcode = str(row[1][4])
    postcode_5 = row[1][5]

    users_df.iloc[index, 5] = str(int(round(postcode_5)))

    if len(postcode) >= 5:
        users_df.iloc[index, 5] = postcode

    if gender >= 0.5:
        users_df.iloc[index, 1] = 'M'
    else:
        users_df.iloc[index, 1] = 'F'

    if age < 18:
        users_df.iloc[index, 2] = 1
    elif 18 <= age < 25:
        users_df.iloc[index, 2] = 18
    elif 25 <= age < 35:
        users_df.iloc[index, 2] = 25
    elif 35 <= age < 45:
        users_df.iloc[index, 2] = 35
    elif 45 <= age < 50:
        users_df.iloc[index, 2] = 45
    elif 50 <= age < 56:
        users_df.iloc[index, 2] = 50
    else:
        users_df.iloc[index, 2] = 56

users_df[['Alder', 'Jobb']] = users_df[['Alder', 'Jobb']].astype(dtype=int)

users_df['Postkode'] = users_df['Postkode_5']
users_df.drop('Postkode_5', axis=1, inplace=True)

print(users_df.head(20))
```

```
    BrukerID Kjonn  Alder  Jobb    Postkode
```

```
0           0     F     45     6       92103
1           1     M     50    16  55405-2546
2           2     M     18    20       44089
3           3     M     35     1       33304
4           4     M     35     6       48105
5           5     M     25    20         664
6           6     M     50    14         379
7           7     F     25     0         264
8           8     M     25     4       70806
9           9     M     25    19       45701
10         10     F     18     1       95864
11         11     M     35     1         478
12         12     M     45     0       10543
13         13     M     50     7       34243
14         14     M     25     4       53140
15         15     F     18     4       60625
16         16     M     25    17       03570
17         17     M     35     7       30117
18         18     M     50     1       01096
19         19     M     25    15       02143
```

[14]:
```python
missing_vals = users_df.isnull().sum()
perc = round(missing_vals / users_df.shape[0] * 100, 2)
print(f'There are {users_df.shape[0]} rows in the dataset.')
print(f'Proportion of missing data for each column in %: \n{perc}')
```

```
There are 6040 rows in the dataset.
Proportion of missing data for each column in %:
BrukerID    0.0
Kjonn       0.0
Alder       0.0
Jobb        0.0
Postkode    0.0
dtype: float64
```

[15]:
```python
users_df.to_csv('cleaned_data/bruker.csv', index=False)
```

[16]:
```python
excel = pd.ExcelFile('raw_data/film.xlsx')
movies_df = excel.parse(sheet_name='film', index_col=None)
movies_df.drop(labels=['Unnamed: 0'], axis=1, inplace=True)
print(f'There are {movies_df.count()[0]} movies in the dataset')
```

```
There are 3883 movies in the dataset
```

[17]:
```python
missing_vals = movies_df.isnull().sum()
perc = round(missing_vals / movies_df.shape[0] * 100, 2)
print(f'Proportion of missing data for each column in %: \n{perc}')
```

```python
duplicate_rows = movies_df[movies_df.duplicated(['FilmID'])]
print(duplicate_rows)
duplicate_rows = movies_df[movies_df.duplicated(['Tittel'])]
print(duplicate_rows)
```

```
Proportion of missing data for each column in %:
FilmID     0.0
Tittel     0.0
Sjanger    0.0
dtype: float64
Empty DataFrame
Columns: [FilmID, Tittel, Sjanger]
Index: []
Empty DataFrame
Columns: [FilmID, Tittel, Sjanger]
Index: []
```

There are no missing or duplicate values in this dataset.

[18]:
```python
genres = ["Action", "Adventure", "Animation", "Children's", "Comedy", "Crime",␣
 ↪"Documentary", "Drama", "Fantasy",
          "Film-Noir", "Horror", "Musical", "Mystery", "Romance", "Sci-Fi",␣
 ↪"Thriller", "War", "Western"]
for genre in genres:
    movies_df.loc[:,genre] = 0
```

[19]:
```python
for row in movies_df.iterrows():
    index = row[0]
    genre_data = row[1][2]

    genres = movies_df.columns.values.tolist()
    genres.remove('FilmID')
    genres.remove('Tittel')
    genres.remove('Sjanger')

    for genre in genres:
        if genre in genre_data:
            movies_df.loc[index, genre] = 1
```

[20]:
```python
movies_df.drop(labels=['Sjanger'], axis=1, inplace=True)
```

[21]:
```python
movies_df.to_csv('cleaned_data/film.csv', index=False)
```

analysis-and-modelling

May 13, 2024

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from math import sqrt
import time
from scipy.sparse import csr_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import minmax_scale

plt.style.use('ggplot')
```

[2]:
```python
ratings_df = pd.read_csv("cleaned_data/rangering.csv")
movies_df = pd.read_csv("cleaned_data/film.csv")
users_df = pd.read_csv("cleaned_data/bruker.csv")
```

[3]:
```python
users_df.head()
```

[3]:

	BrukerID	Kjonn	Alder	Jobb	Postkode
0	0	F	45	6	92103
1	1	M	50	16	55405-2546
2	2	M	18	20	44089
3	3	M	35	1	33304
4	4	M	35	6	48105

[4]:
```python
users_df.describe()
```

[4]:

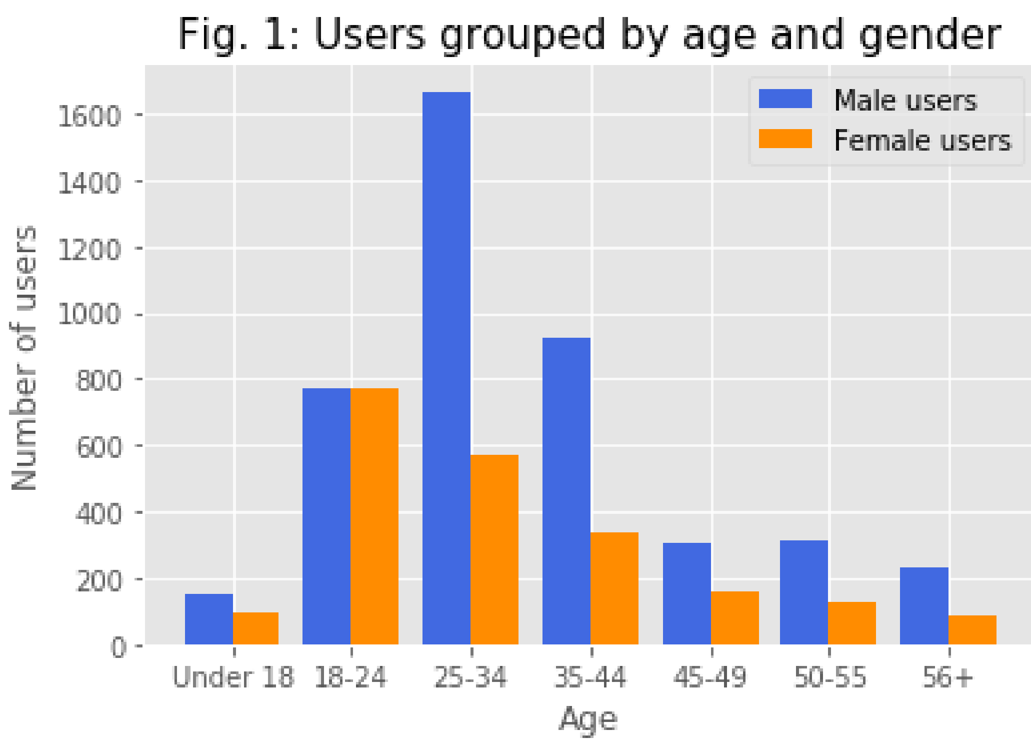
	BrukerID	Alder	Jobb
count	6040.000000	6040.000000	6040.000000
mean	3020.465894	29.894868	8.200828

```
std    1743.799216   12.484335   10.933051
min       0.000000    1.000000    0.000000
25%    1510.750000   25.000000    1.000000
50%    3020.500000   25.000000    6.000000
75%    4530.250000   35.000000   13.000000
max    6040.000000   56.000000   98.000000
```

[5]:

```
user_age_gender_data = users_df[['Kjonn', 'Alder']]
female_user_ages = user_age_gender_data.loc[user_age_gender_data['Kjonn'] ==␣
 ↪'F'].sort_values('Alder')
male_user_ages = user_age_gender_data.loc[user_age_gender_data['Kjonn'] == 'M'].
 ↪sort_values('Alder')


def get_group_count(min_age, max_age, dataset):
    age_group = dataset.apply(lambda x: True if max_age > x['Alder'] > min_age␣
 ↪else False, axis=1)
    count = len(age_group[age_group == True].index)
    return count


G1_male = get_group_count(0, 18, male_user_ages)
G2_male = get_group_count(17, 25, male_user_ages)
G3_male = get_group_count(24, 35, male_user_ages)
G4_male = get_group_count(34, 45, male_user_ages)
G5_male = get_group_count(44, 50, male_user_ages)
G6_male = get_group_count(49, 56, male_user_ages)
G7_male = get_group_count(55, 200, male_user_ages)

G1_female = get_group_count(0, 18, female_user_ages)
G2_female = get_group_count(17, 25, female_user_ages)
G3_female = get_group_count(24, 35, female_user_ages)
G4_female = get_group_count(34, 45, female_user_ages)
G5_female = get_group_count(44, 50, female_user_ages)
G6_female = get_group_count(49, 56, female_user_ages)
G7_female = get_group_count(55, 200, female_user_ages)

labels = ['Under 18', '18-24', '25-34', '35-44', '45-49', '50-55', '56+']
men_grouped = [G1_male, G2_male, G3_male, G4_male, G5_male, G6_male, G7_male]
women_grouped = [G1_female, G2_male, G3_female, G4_female, G5_female,␣
 ↪G6_female, G7_female]
x = np.arange(len(labels))
width = 0.40

fig1, ax1 = plt.subplots()
rects1 = ax1.bar(x - width/2, men_grouped, width, label='Male users',␣
 ↪color='royalblue')
```

```
rects2 = ax1.bar(x + width/2, women_grouped, width, label='Female users',␣
 ↪color='darkorange')

ax1.set_ylabel('Number of users', size=12)
ax1.set_xlabel('Age', size=12)
ax1.set_xticks(x)
ax1.set_xticklabels(labels)
ax1.set_title('Fig. 1: Users grouped by age and gender', size=15)
ax1.legend()

plt.show()
```



[6]:
```
movies_df.head()
```

[6]:
```
   FilmID                                   Tittel  Action  Adventure  \
0       0                    Autumn in New York (2000)       0          0
1       1  Vie est belle, La (Life is Rosey) (1987)       0          0
2       2                       Defying Gravity (1997)       0          0
3       3                      Ruthless People (1986)       0          0
4       4                     Portraits Chinois (1996)       0          0

   Animation  Children's  Comedy  Crime  Documentary  Drama  Fantasy  \
0          0           0       0      0            0      1        0
```

```
1           0            0       1      0             0      1         0
2           0            0       0      0             0      1         0
3           0            0       1      0             0      0         0
4           0            0       0      0             0      1         0

   Film-Noir  Horror  Musical  Mystery  Romance  Sci-Fi  Thriller  War  \
0          0       0        0        0        1       0         0    0
1          0       0        0        0        0       0         0    0
2          0       0        0        0        0       0         0    0
3          0       0        0        0        0       0         0    0
4          0       0        0        0        0       0         0    0

   Western
0        0
1        0
2        0
3        0
4        0
```

[7]:
```python
movies_df.describe()
```

[7]:
```
            FilmID       Action    Adventure    Animation   Children's  \
count  3883.000000  3883.000000  3883.000000  3883.000000  3883.000000
mean   1973.687098     0.129539     0.072882     0.027041     0.040948
std    1142.105375     0.335839     0.259976     0.162224     0.198195
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%     985.500000     0.000000     0.000000     0.000000     0.000000
50%    1973.000000     0.000000     0.000000     0.000000     0.000000
75%    2963.500000     0.000000     0.000000     0.000000     0.000000
max    3952.000000     1.000000     1.000000     1.000000     1.000000

            Comedy        Crime  Documentary        Drama      Fantasy  \
count  3883.000000  3883.000000  3883.000000  3883.000000  3883.000000
mean      0.309039     0.054339     0.032707     0.412825     0.017512
std       0.462157     0.226715     0.177891     0.492405     0.131187
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       1.000000     0.000000     0.000000     1.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

         Film-Noir       Horror      Musical      Mystery      Romance  \
count  3883.000000  3883.000000  3883.000000  3883.000000  3883.000000
mean      0.011074     0.088334     0.029359     0.027041     0.121040
std       0.104662     0.283816     0.168832     0.162224     0.326216
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
```

```
50%        0.000000     0.000000     0.000000     0.000000     0.000000
75%        0.000000     0.000000     0.000000     0.000000     0.000000
max        1.000000     1.000000     1.000000     1.000000     1.000000

            Sci-Fi     Thriller          War      Western
count  3883.000000  3883.000000  3883.000000  3883.000000
mean      0.071079     0.126706     0.036827     0.017512
std       0.256990     0.332686     0.188362     0.131187
min       0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000
```

[8]:

```python
ratings_df.head()
```

[8]:

	BrukerID	FilmID	Rangering	Tidstempel
0	0	1561	4	959441640
1	0	1540	3	959441640
2	0	88	3	959441640
3	0	620	4	959441640
4	0	3771	5	959442113

[9]:

```python
ratings_df.describe()
```

[9]:

	BrukerID	FilmID	Rangering	Tidstempel
count	900117.000000	900117.000000	900117.000000	9.001170e+05
mean	2991.868675	1989.666242	3.581491	9.722407e+08
std	1736.208380	1126.359558	1.117162	1.214212e+07
min	0.000000	0.000000	1.000000	9.567039e+08
25%	1458.000000	1037.000000	3.000000	9.653030e+08
50%	2967.000000	1959.000000	4.000000	9.730170e+08
75%	4501.000000	2963.000000	4.000000	9.752206e+08
max	6040.000000	3952.000000	5.000000	1.046455e+09

[10]:

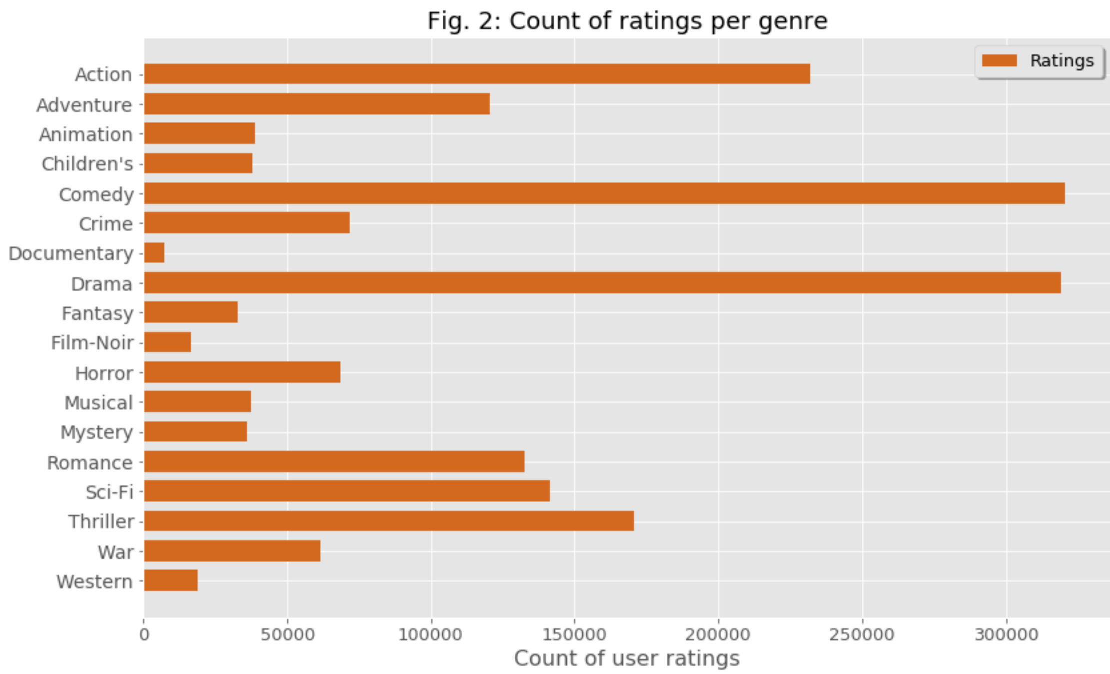
```python
genres = list(movies_df)[2:]
merged = pd.merge(ratings_df, movies_df, on='FilmID')

cleaned = merged[['FilmID', 'BrukerID', 'Rangering', 'Action', 'Adventure',␣
 ↪'Animation', "Children's", 'Comedy',
                  'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir',␣
 ↪'Horror', 'Musical',
                  'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western',␣
 ↪'Tittel']]

movie_ratings = cleaned.sort_values('FilmID')
rating_scale = [1, 2, 3, 4, 5]
```

```
genres_rating_count = []
for genre in genres:
    for rating in rating_scale:
        genre_list = movie_ratings.loc[movie_ratings[genre] == 1]
        genre_rated_count = len(genre_list.loc[genre_list['Rangering'] ==␣
 ↪rating])
        genres_rating_count.append([genre, rating, genre_rated_count])



def get_rating_count(score):
    rating_count = []
    for count in genres_rating_count:
        if count[1] == score:
            rating_count.append(count[2])

    return rating_count



def get_total_rating_count():
    all_ratings = []
    totals = []
    for count in genres_rating_count:
        all_ratings.append(count[2])

    lower = 0
    for i in range(lower, len(all_ratings), 5):
        lower = i
        totals.append(sum(all_ratings[lower:lower+5]))

    return totals


genre_df = pd.DataFrame({'Genre': [genre for genre in genres], 'Rated 1':␣
 ↪get_rating_count(1), 'Rated 2': get_rating_count(2),
              'Rated 3': get_rating_count(3), 'Rated 4': get_rating_count(4),␣
 ↪'Rated 5': get_rating_count(5),
              'Total nr of ratings': get_total_rating_count()})
genre_df["Average rating"] = (genre_df["Rated 1"] + (genre_df["Rated 2"] * 2) +␣
 ↪(genre_df["Rated 3"] * 3) +
                             (genre_df["Rated 4"] * 4) + (genre_df["Rated 5"] *␣
 ↪5)) / genre_df['Total nr of ratings']
genre_df
```

[10]:
```
           Genre  Rated 1  Rated 2  Rated 3  Rated 4  Rated 5  \
0         Action    14884    28359    63691    77911    46969
1      Adventure     7675    15038    34070    39809    24009
2      Animation     1886     3264     9871    14258     9719
3     Children's     3500     4776    10932    11917     6554
4         Comedy    19410    36985    87148   111126    66128
5          Crime     2879     6726    17580    25579    18800
6    Documentary      246      425     1293     2723     2405
7          Drama    10917    26252    76415   118548    86967
8        Fantasy     2152     4269     9405    10513     6291
9      Film-Noir      243      764     2877     6186     6358
10        Horror     8040    10785    19195    19735    10990
11       Musical     1868     3399     9471    13113     9448
12       Mystery     1563     3588     8994    12740     8956
13       Romance     5710    13734    36372    48137    28794
14        Sci-Fi     9908    18189    38582    45602    29229
15      Thriller     9103    19085    45076    60132    37376
16           War     2208     4393    12300    21665    21060
17       Western      958     1757     4833     6634     4452

    Total nr of ratings  Average rating
0                231814        3.490574
1                120601        3.476273
2                 38998        3.683625
3                 37679        3.351628
4                320797        3.522377
5                 71564        3.708387
6                  7092        3.932882
7                319099        3.765894
8                 32630        3.445051
9                 16428        4.074507
10                68745        3.216016
11                37299        3.666881
12                35841        3.667894
13               132747        3.606952
14               141510        3.466787
15               170772        3.571481
16                61626        3.892091
17                18634        3.636739
```

[11]:
```python
total_ratings = genre_df.iloc[:, 6].values.tolist()
fig2, ax2 = plt.subplots()
fig2_labels = [genre for genre in genres]
x = np.arange(len(fig2_labels))
width = 0.7
ax2.barh(x, total_ratings, width, label='Ratings', color='chocolate')
```

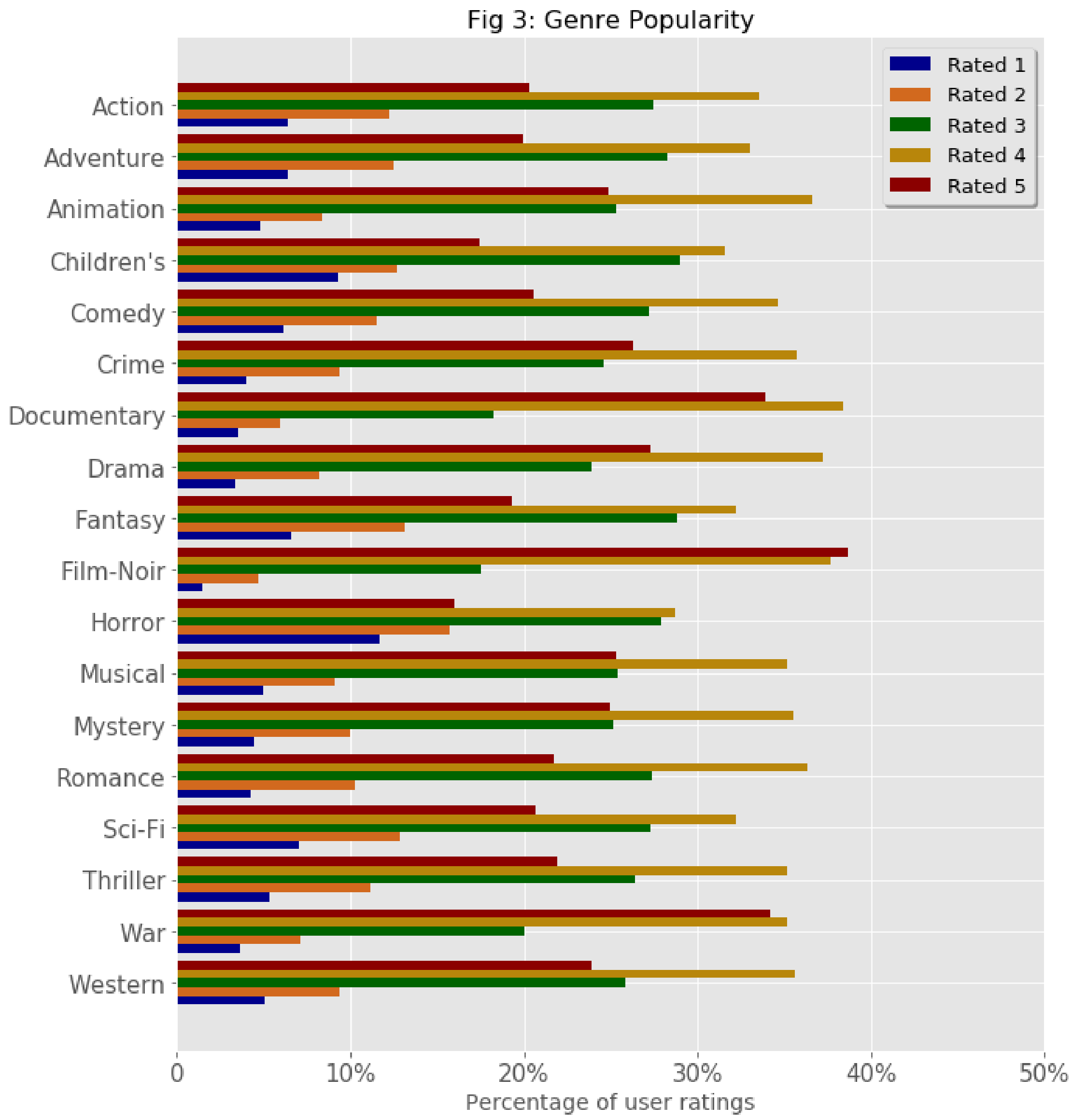
```
ax2.set_xlabel('Count of user ratings', size=16)
ax2.set_yticks(x)
ax2.set_yticklabels(fig2_labels, fontdict={'fontsize': 14})
ax2.set_title('Fig. 2: Count of ratings per genre', size=18)
ax2.invert_yaxis()
ax2.legend(shadow=0.4, prop={"size": 13})
plt.xticks(fontsize=13)


fig2.set_figheight(8)
fig2.set_figwidth(13)
plt.show()
```



[12]:
```
def get_rating_percentages(score):
    rated_count = genre_df.iloc[:, score].values.tolist()
    rating_percentages = []
    i = 0

    for total in total_ratings:
        percentage = (100 / total) * rated_count[i]
        i += 1
        rating_percentages.append(round(percentage, 1))

    return rating_percentages
```

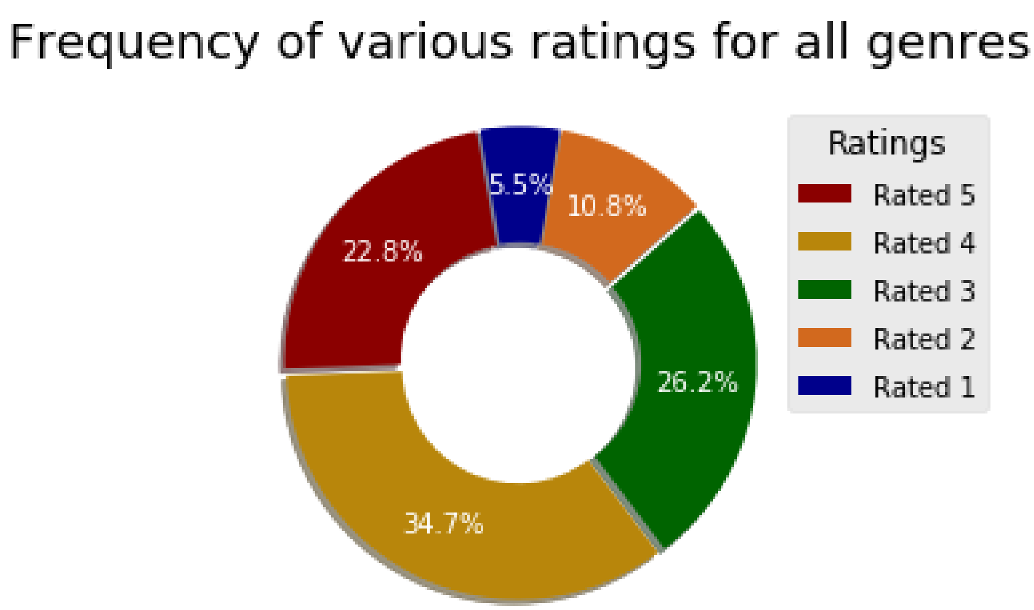
```python
genre_rating_percentage_data = {'Genre': [genre for genre in genres], 'Rated 1':
 ↪ get_rating_percentages(1),
                                'Rated 2': get_rating_percentages(2), 'Rated 3':
 ↪ get_rating_percentages(3),
                                'Rated 4': get_rating_percentages(4), 'Rated 5':
 ↪ get_rating_percentages(5),
                                'Total nr of ratings': get_total_rating_count()}
genre_rating_percent_df = pd.DataFrame(genre_rating_percentage_data)

rated_1_percentage = genre_rating_percent_df.iloc[:, 1].values.tolist()
rated_2_percentage = genre_rating_percent_df.iloc[:, 2].values.tolist()
rated_3_percentage = genre_rating_percent_df.iloc[:, 3].values.tolist()
rated_4_percentage = genre_rating_percent_df.iloc[:, 4].values.tolist()
rated_5_percentage = genre_rating_percent_df.iloc[:, 5].values.tolist()

fig3, ax3 = plt.subplots()
fig3_ylabels = [genre for genre in genres]
fig3_xlabels = ['0', '10%', '20%', '30%', '40%', '50%']
x = np.arange(len(fig3_ylabels))
width = 0.17

ax3.barh(x + width*2, rated_1_percentage, width, label='Rated 1',␣
 ↪color='darkblue')
ax3.barh(x + width, rated_2_percentage, width, label='Rated 2',␣
 ↪color='chocolate')
ax3.barh(x, rated_3_percentage, width, label='Rated 3', color='darkgreen')
ax3.barh(x - width, rated_4_percentage, width, label='Rated 4',␣
 ↪color='darkgoldenrod')
ax3.barh(x - width*2, rated_5_percentage, width, label='Rated 5',␣
 ↪color='darkred')


ax3.set_xlabel('Percentage of user ratings', size=14)
ax3.set_xticklabels(fig3_xlabels, fontdict={'fontsize': 15})
ax3.set_xlim(right=50)
ax3.set_yticks(x)
ax3.set_yticklabels(fig3_ylabels, fontdict={'fontsize': 15})
ax3.set_title('Fig 3: Genre Popularity', size=16)
ax3.invert_yaxis()
ax3.legend(shadow=0.4, prop={"size": 13})


fig3.set_figheight(12)
fig3.set_figwidth(10)
```

```
plt.show()
```



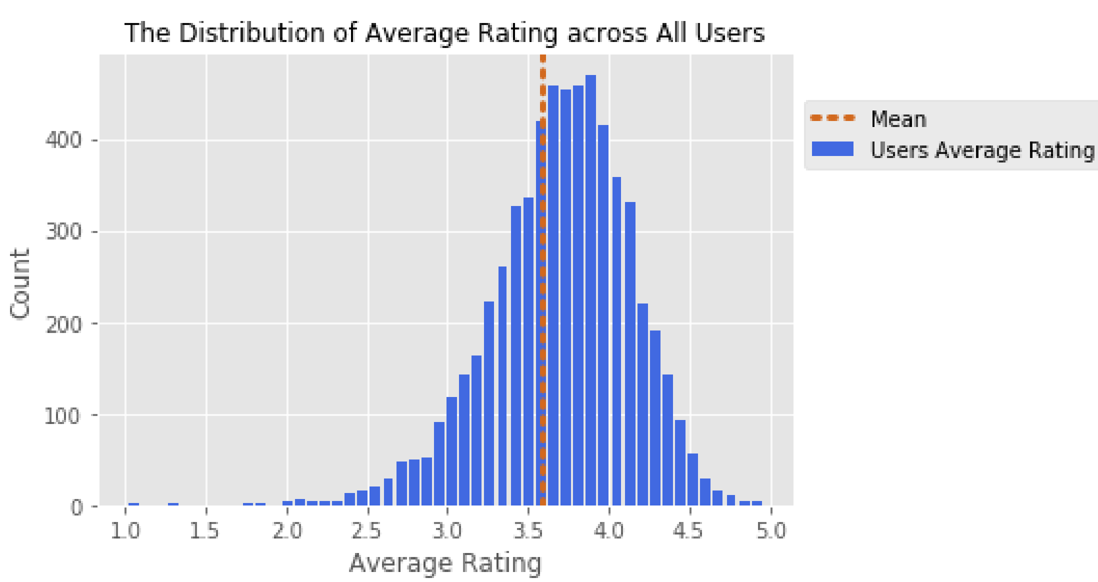
[13]:
```
def get_rating_percentage_total(score):
    rated_count = genre_df.iloc[:, score].values.tolist()
    total_ratings_count = 0
    sum_ratings = 0

    for count in rated_count:
        sum_ratings += count

    for total in total_ratings:
        total_ratings_count += total
```

```
    return round((sum_ratings / total_ratings_count) * 100, 1)


ratings_data = [get_rating_percentage_total(5), get_rating_percentage_total(4), get_rating_percentage_total(3),
                get_rating_percentage_total(2), get_rating_percentage_total(1)]


labels = ["Rated 5", "Rated 4", "Rated 3", "Rated 2", "Rated 1"]
fig4, ax4 = plt.subplots(subplot_kw=dict(aspect="equal"))
wedges, texts, autotext = ax4.pie(ratings_data, autopct='%1.1f%%', pctdistance=0.75, shadow=True,
                                  wedgeprops=dict(width=0.5), startangle=100, textprops=dict(color="w"),
                                  explode=(0.02, 0.02, 0.02, 0.02, 0.02), colors=['darkred', 'darkgoldenrod',
                                                                                 'darkgreen', 'chocolate', 'darkblue'])

ax4.legend(wedges, labels, loc="upper right", bbox_to_anchor=(1.0, 0.05, 0.33, 0.9), title='Ratings',
           title_fontsize='12', labelspacing=0.8)
ax4.set_title("Frequency of various ratings for all genres", size=18)

plt.show()
```

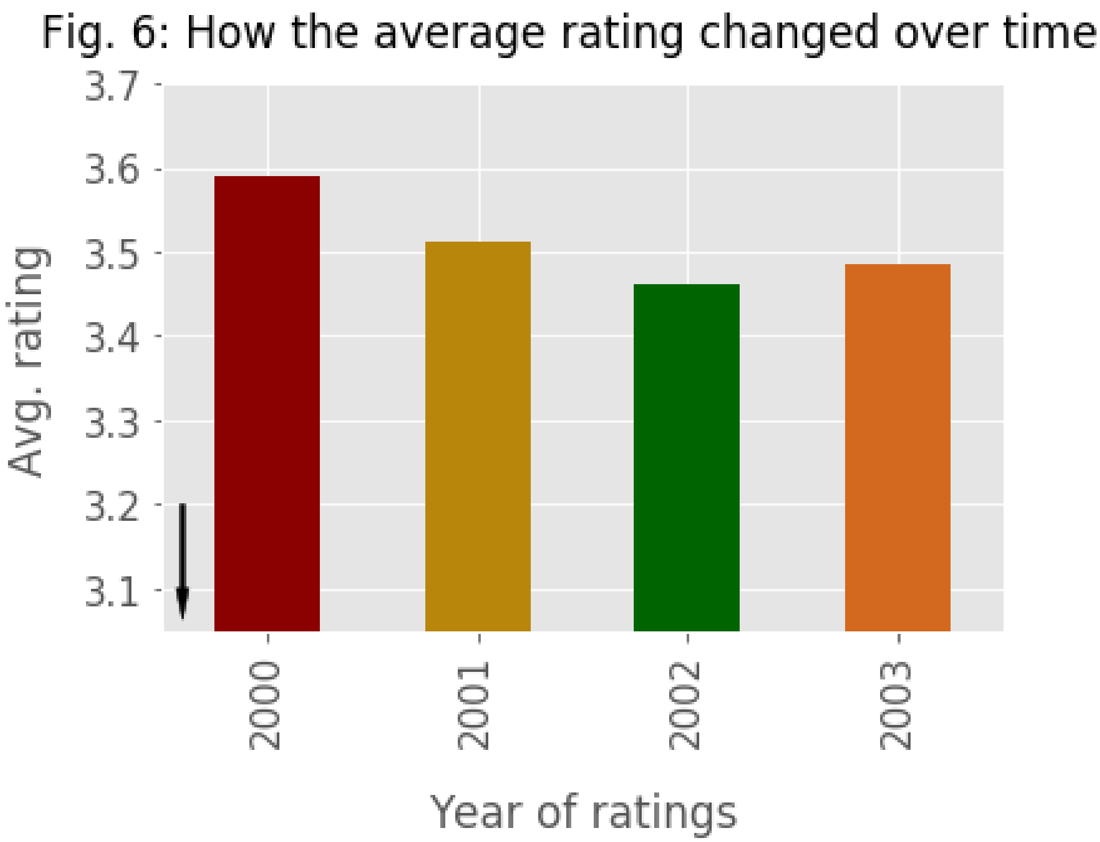


[14]:
```python
users_avg_rating = pd.DataFrame(ratings_df.groupby('BrukerID')['Rangering'].
    mean())
print(users_avg_rating.head())

users_avg_rating = users_avg_rating['Rangering']
fig5, ax5 = plt.subplots()
n, bins, patches = ax5.hist(users_avg_rating, label='Users Average Rating',
                            stacked=True, color='royalblue', bins=50, rwidth=0.
    8)

ax5.set_xlabel('Average Rating', size=12)
ax5.set_ylabel('Count', size=12)
ax5.set_title('The Distribution of Average Rating across All Users', size=12)

plt.axvline(ratings_df["Rangering"].mean(), color='chocolate',
    linestyle='dotted', dash_capstyle="round",
            linewidth=3, label="Mean")
ax5.legend(bbox_to_anchor=(1, 0.92))

plt.show()
```

```
          Rangering
BrukerID
0          3.296610
1          3.487179
2          3.600000
3          4.000000
4          3.891892
```



[15]:
```python
def convert_timestamp_to_year(timestamp):
    date_string = time.ctime(timestamp)
    tokens = date_string.split(sep=" ")
    year = int(tokens[-1])
    return year


ratings_copy = ratings_df.copy()
ratings_copy['Rangeringsår'] = ratings_df["Tidstempel"].map(lambda timestamp:␣
 ↪convert_timestamp_to_year(timestamp))
yearly_mean_rating = ratings_copy.groupby('Rangeringsår')['Rangering'].mean()


plt.figure()
plt.rcParams.update({'font.size': 14})
yearly_mean_rating.plot(kind='bar', figsize=(10, 10), title="Fig. 6: How the␣
 ↪average rating changed over time",
                        subplots=True, color=['darkred', 'darkgoldenrod',␣
 ↪'darkgreen', 'chocolate'],
                        ylim=(3.05, 3.7), fontsize=15, label='')
plt.xlabel(xlabel='Year of ratings', labelpad=18)
plt.ylabel(ylabel='Avg. rating', labelpad=12)


plt.arrow(x=(-0.4), y=3.2, dx=0, dy=(-0.1), width=0.02,␣
 ↪length_includes_head=False, head_length=0.037, head_width=0.06,
          color='black')

plt.show()
```



[16]:
```
X = ratings_df.drop(columns='Rangering')
y = ratings_df["Rangering"].values


X_train, X_val_and_test, y_train, y_val_and_test = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=101)
X_val, X_test, y_val, y_test = train_test_split(X_val_and_test, y_val_and_test,␣
 ↪test_size=0.5, random_state=101)


train_df = X_train.copy()
train_df["Rangering"] = y_train

train_df
```

[16]:

	BrukerID	FilmID	Tidstempel	Rangering
708938	4341	1210	976573899	3
371257	1425	3192	970774826	4
845603	5565	2299	976239652	5
667558	3970	83	965770467	3

```
821518      5320    3843    976249576         4
…             …      …       …         …
661055      3915    1846    974698675         4
204614        32     884    974679433         4
476497      2357     101    975126955         4
214539       122    2098    974662091         4
176991      5383    2759    962992451         5

[630081 rows x 4 columns]
```

[17]:
```python
baseline_y_pred = pd.DataFrame(train_df.groupby('FilmID')['Rangering'].mean())


val_movies_dict = {'FilmID': X_val["FilmID"], 'Actual rating': y_val}
val_movies_df = pd.DataFrame(val_movies_dict)

y_pred_and_y_true = pd.merge(baseline_y_pred, val_movies_df, on='FilmID')
baseline_y_pred_vs_y_true = y_pred_and_y_true.rename(columns={"Rangering":␣
 ↪"Predicted rating"})

baseline_y_pred_vs_y_true
```

[17]:
```
        FilmID  Predicted rating  Actual rating
0            0          2.055556              1
1            0          2.055556              4
2            0          2.055556              3
3            0          2.055556              2
4            0          2.055556              2
…          …                 …             …
134982    3952          2.353846              2
134983    3952          2.353846              1
134984    3952          2.353846              1
134985    3952          2.353846              3
134986    3952          2.353846              3

[134987 rows x 3 columns]
```

[18]:
```python
print("RMSE baseline model: ",␣
 ↪sqrt(mean_squared_error(baseline_y_pred_vs_y_true["Predicted rating"],
                                                              ␣
 ↪baseline_y_pred_vs_y_true["Actual rating"])))
```

```
RMSE baseline model:  0.981626106076701
```

[19]:
```python
content_train_df = pd.merge(train_df, movies_df, on='FilmID')
content_train_df.drop(columns=['Tidstempel', 'FilmID', 'Tittel'], inplace=True)␣
 ↪ # Remove useless features
```

```
content_train_df
```

[19]:

	BrukerID	Rangering	Action	Adventure	Animation	Children's	Comedy
0	4341	3	0	0	0	0	0
1	1242	3	0	0	0	0	0
2	1056	5	0	0	0	0	0
3	4381	4	0	0	0	0	0
4	1134	4	0	0	0	0	0
…	…	…	…	…	…	…	…
630076	4551	2	0	0	0	0	0
630077	455	4	0	0	0	0	0
630078	2305	4	1	0	0	0	0
630079	317	3	0	0	0	0	0
630080	1891	4	0	0	0	0	0

	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	1	0	0	0	0
…	…	…	…	…	…	…	…
630076	0	0	0	0	0	0	0
630077	1	0	1	0	0	0	0
630078	0	0	1	0	0	0	0
630079	0	0	1	0	0	0	0
630080	0	0	0	0	1	1	0

	Mystery	Romance	Sci-Fi	Thriller	War	Western
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
…	…	…	…	…	…	…
630076	0	0	0	1	0	0
630077	0	0	0	0	0	0
630078	0	0	0	1	0	0
630079	0	0	0	0	0	0
630080	0	0	0	0	0	0

[630081 rows x 20 columns]

[20]:
```
y_grouped_by_user = content_train_df.groupby(["BrukerID"])
y_train_listed = []
```

```python
for i, j in y_grouped_by_user:
    y_train_listed.append(j["Rangering"].values)
    y_train_listed[0]
```

[20]:
```
array([4, 1, 4, 3, 5, 3, 4, 3, 5, 2, 4, 4, 4, 3, 4, 5, 2, 2, 5, 4, 3, 2,
       4, 4, 3, 2, 3, 3, 5, 4, 5, 1, 5, 3, 3, 2, 4, 3, 3, 4, 5, 3, 2, 4,
       3, 5, 1, 1, 4, 4, 3, 2, 1, 4, 4, 3, 3, 3, 4, 4, 3, 3, 1, 4, 4, 5,
       4, 3, 3, 4, 2, 3, 2, 2, 4, 4, 3, 5, 4, 2, 4, 3, 3, 3, 4, 2, 4, 5,
       4, 3, 3, 4, 3, 2, 3, 2, 3, 4, 3, 4, 4, 4, 5, 3, 3, 4, 3, 3, 4, 3,
       3, 3, 3, 3, 3, 3, 4, 4, 3, 4, 4, 4, 3, 4, 3, 3, 5, 3, 4, 2, 3, 3,
       3, 3, 5, 2, 4, 3, 3, 4, 4, 3, 3, 5, 3, 2, 1, 3, 3, 3, 2, 3, 4, 3,
       2, 3, 3, 3, 4, 3, 4, 4, 2, 3, 3, 3, 2, 3, 4, 3], dtype=int64)
```

[21]:
```python
content_train_df.drop(columns='Rangering', inplace=True)
x_grouped_by_user = content_train_df.groupby(["BrukerID"])
x_train_listed = []

for user_id, group in x_grouped_by_user:
    x_train_listed.append(group.drop(columns='BrukerID'))

x_train_listed[0]
```

[21]:

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	\
2824	1	1	0	0	0	0	0	
9073	0	0	0	0	1	0	0	
14756	1	1	0	0	0	0	0	
18316	0	0	0	0	1	0	0	
22117	0	0	1	1	0	0	0	
…	…	…	…	…	…	…	…	…
618870	0	0	0	0	1	0	0	
618897	0	0	0	0	0	0	1	
619528	0	0	0	0	0	0	0	
625089	0	0	0	0	1	0	0	
626018	0	0	1	0	0	0	0	

	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	\
2824	0	0	0	0	0	0	0	1	
9073	0	0	0	0	0	0	0	1	
14756	0	0	0	0	0	0	0	1	
18316	0	0	0	0	0	0	1	0	
22117	0	0	0	0	1	0	0	0	
…	…	…	…	…	…	…	…	…	…
618870	0	0	0	0	0	0	1	0	
618897	0	0	0	0	0	0	0	0	
619528	1	0	0	0	0	0	1	0	
625089	0	0	0	0	0	0	1	0	
626018	0	0	0	0	0	0	0	0	

```
        Thriller  War  Western
2824           1    0        0
9073           0    0        1
14756          0    0        0
18316          0    0        0
22117          0    0        0
…          …   …       …
618870         0    0        0
618897         0    0        0
619528         0    1        0
625089         0    0        0
626018         0    0        0

[170 rows x 18 columns]
```

[22]:
```python
all_movies = movies_df.drop(columns=['Tittel', 'FilmID'])
all_movies
```

[22]:
```
      Action  Adventure  Animation  Children's  Comedy  Crime  Documentary  \
0          0          0          0           0       0      0            0
1          0          0          0           0       1      0            0
2          0          0          0           0       0      0            0
3          0          0          0           0       1      0            0
4          0          0          0           0       0      0            0
…       …         …         …          …      …    …          …
3878       0          0          0           0       0      0            0
3879       0          0          0           0       0      0            1
3880       0          0          0           0       1      0            0
3881       1          1          0           0       1      0            0
3882       0          0          0           0       0      0            0

      Drama  Fantasy  Film-Noir  Horror  Musical  Mystery  Romance  Sci-Fi  \
0         1        0          0       0        0        0        1       0
1         1        0          0       0        0        0        0       0
2         1        0          0       0        0        0        0       0
3         0        0          0       0        0        0        0       0
4         1        0          0       0        0        0        0       0
…     …      …        …        …     …      …       …       …
3878      0        0          0       1        0        0        0       0
3879      0        0          0       0        0        0        0       0
3880      0        0          0       1        0        0        0       0
3881      0        0          0       0        0        0        1       0
3882      0        0          0       1        0        0        0       0

      Thriller  War  Western
0            0    0        0
```

```
1              0    0        0
2              0    0        0
3              0    0        0
4              0    0        0
…           …  …       …
3878           0    0        0
3879           0    0        0
3880           0    0        0
3881           0    0        0
3882           0    0        0

[3883 rows x 18 columns]
```

[23]:
```python
user_ids = []
for user_id, group in x_grouped_by_user:
    user_ids.append(user_id)

movie_ids = movies_df["FilmID"].values

df_val = X_val.copy()
df_val["Rangering"] = y_val
validation_matrix = pd.DataFrame(index=user_ids, columns=movie_ids)
for array in df_val.to_records():
    user = array['BrukerID']
    movie = array['FilmID']
    true_rating = array['Rangering']
    validation_matrix.loc[user][movie] = true_rating

validation_matrix
```

[23]:
```
      0    1    2    3    4    5    6    7    8    9    … 3943 3944 3945  \
0     NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
1     NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
2     NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
3     NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
4     NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
…    …   …   …   …   …   …   …   …   …   …   …  …   …   …   …
6036  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
6037  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
6038  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
6039  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN
6040  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  …  NaN  NaN  NaN

     3946 3947 3948 3949 3950 3951 3952
0     NaN  NaN  NaN  NaN  NaN  NaN  NaN
1     NaN  NaN  NaN  NaN  NaN  NaN  NaN
2     NaN  NaN  NaN  NaN  NaN  NaN  NaN
```

```
3     NaN  NaN  NaN  NaN  NaN  NaN  NaN
4     NaN  NaN  NaN  NaN  NaN    4  NaN
…    …    …    …    …    …    …    …
6036  NaN  NaN  NaN  NaN  NaN  NaN  NaN
6037  NaN  NaN  NaN  NaN  NaN  NaN  NaN
6038  NaN  NaN  NaN  NaN  NaN  NaN  NaN
6039  NaN  NaN  NaN  NaN  NaN  NaN  NaN
6040  NaN  NaN  NaN  NaN  NaN  NaN  NaN

[6040 rows x 3883 columns]
```

[24]:

```
ml_algorithms = {"Linear regression": LinearRegression(), "Lasso":␣
↪Lasso(alpha=1.0, max_iter=10000),
                 "KNN_7": KNeighborsRegressor(n_neighbors=7),
                 "RFR": RandomForestRegressor(n_estimators=1000, n_jobs=3,␣
↪max_features="auto", random_state=0),
                 "SVR": SVR(C=1.0)}

CBF_models_listed = []
RMSE_CBF_listed = []

for name, ml_alg in ml_algorithms.items():

    CBF_predictions = []


    for i, x in enumerate(x_train_listed):

        ml_alg.fit(x_train_listed[i], y_train_listed[i])

        prediction = ml_alg.predict(all_movies)
        prediction = np.clip(prediction, 1, 5)

        CBF_predictions.append(prediction)

    df_predict = pd.DataFrame(CBF_predictions, index=user_ids,␣
↪columns=movie_ids)

    num_actual = validation_matrix.to_numpy().flatten()[validation_matrix.
↪notna().to_numpy().flatten()]
    num_predict = df_predict.to_numpy().flatten()[validation_matrix.notna().
↪to_numpy().flatten()]

    RMSE_CBF_listed.append(sqrt(mean_squared_error(num_predict, num_actual)))
    CBF_models_listed.append(name)
```

```
RMSE_CBF_df = pd.DataFrame({"Model": CBF_models_listed, "RMSE":␣
↪RMSE_CBF_listed})
print("RMSE of different content-based filtering models without the year of␣
↪release feature:")
RMSE_CBF_df
```

```
C:\Users\sebas\anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.0, tolerance: 0.0
  positive)
```

```
RMSE of different content-based filtering models without the year of release
feature
```

[24]:

	Model	RMSE
0	Linear regression	1.071366
1	Lasso	1.037082
2	KNN_7	1.061543
3	RFR	1.080374
4	SVR	1.054499

[25]:

```
model = Lasso(alpha=1.0, max_iter=10000)
CBF_predictions = []

for i, j in enumerate(x_train_listed):
    model.fit(x_train_listed[i], y_train_listed[i])
    prediction = model.predict(all_movies)
    prediction = np.clip(prediction, 1, 5)
    CBF_predictions.append(prediction)

CBF_model = pd.DataFrame(CBF_predictions, index=user_ids, columns=movie_ids)
```

```
C:\Users\sebas\anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.0, tolerance: 0.0
  positive)
```

[26]:

```
train_df.head()
```

[26]:

	BrukerID	FilmID	Tidstempel	Rangering
708938	4341	1210	976573899	3
371257	1425	3192	970774826	4
845603	5565	2299	976239652	5
667558	3970	83	965770467	3
821518	5320	3843	976249576	4

```
[ ]: user_matrix = train_df.pivot(index='BrukerID', columns='FilmID',␣
     ↪values='Rangering')

     user_matrix = user_matrix.sub(user_matrix.mean(axis=1), axis=0)

     user_matrix = user_matrix.fillna(0.0)
```

```
[27]: user_dist_matrix = 1 - user_matrix.T.corr()
      user_dist_matrix
```

```
[27]: BrukerID      0         1         2         3         4         5     \
      BrukerID
      0          0.000000  1.008851  1.000000  0.870659  1.004774  1.005061
      1          1.008851  0.000000  1.000000  1.000000  1.038525  1.215623
      2          1.000000  1.000000  0.000000  1.000000  0.954233  1.000000
      3          0.870659  1.000000  1.000000  0.000000  1.000000  1.000000
      4          1.004774  1.038525  0.954233  1.000000  0.000000  0.966535
      …              …         …         …         …         …         …
      6036       1.004525  1.000000  0.914610  1.000000  1.020158  1.000000
      6037       0.987334  1.008252  1.084354  1.001476  0.936568  1.000000
      6038       1.019618  1.011828  1.000000  1.000000  1.076701  1.000000
      6039       1.108225  0.992498  0.989900  1.000000  1.008673  1.029731
      6040       0.992078  1.027741  1.029691  1.071963  1.018350  1.030326

      BrukerID      6         7         8         9     …     6031      6032  \
      BrukerID                                        …
      0          0.985787  0.993054  0.936473  1.000721  …  1.000000  0.993009
      1          1.013271  0.981892  0.966971  0.972867  …  1.000000  1.014202
      2          1.002765  0.967496  1.024260  0.956310  …  1.000000  0.939836
      3          1.000000  1.000000  0.911999  1.033222  …  1.000000  1.096116
      4          1.079381  0.970525  0.924106  0.932877  …  0.994337  0.952138
      …              …         …         …         …    …     …         …
      6036       1.001341  0.931767  0.978402  0.988251  …  0.948502  0.975727
      6037       1.003874  1.016693  0.986043  1.021241  …  1.000000  0.928262
      6038       0.978866  1.000000  1.028991  0.961786  …  1.000000  0.981633
      6039       0.996649  1.000000  0.978282  0.954219  …  1.000000  0.977534
      6040       0.989035  0.981804  0.975827  0.979975  …  0.994760  0.980172

      BrukerID     6033      6034      6035      6036      6037      6038  \
      BrukerID
      0          0.919575  1.008068  1.000520  1.004525  0.987334  1.019618
      1          0.986063  0.993568  1.000000  1.000000  1.008252  1.011828
      2          0.949094  0.938041  0.968789  0.914610  1.084354  1.000000
      3          1.000000  0.998262  1.000000  1.000000  1.001476  1.000000
      4          0.979018  1.007612  1.046063  1.020158  0.936568  1.076701
      …              …         …         …         …         …         …
      6036       0.992479  0.984205  0.964859  0.000000  1.067465  1.000000
```

```
6037      1.027527  0.927628  1.105656  1.067465  0.000000  1.028313
6038      1.010349  0.987804  1.002550  1.000000  1.028313  0.000000
6039      0.981483  0.989847  0.951471  0.941595  1.000000  1.000000
6040      0.961100  1.000266  0.927586  1.008256  0.969143  0.998009

BrukerID      6039      6040
BrukerID
0         1.108225  0.992078
1         0.992498  1.027741
2         0.989900  1.029691
3         1.000000  1.071963
4         1.008673  1.018350
…              …         …
6036      0.941595  1.008256
6037      1.000000  0.969143
6038      1.000000  0.998009
6039      0.000000  0.986514
6040      0.986514  0.000000

[6040 rows x 6040 columns]
```

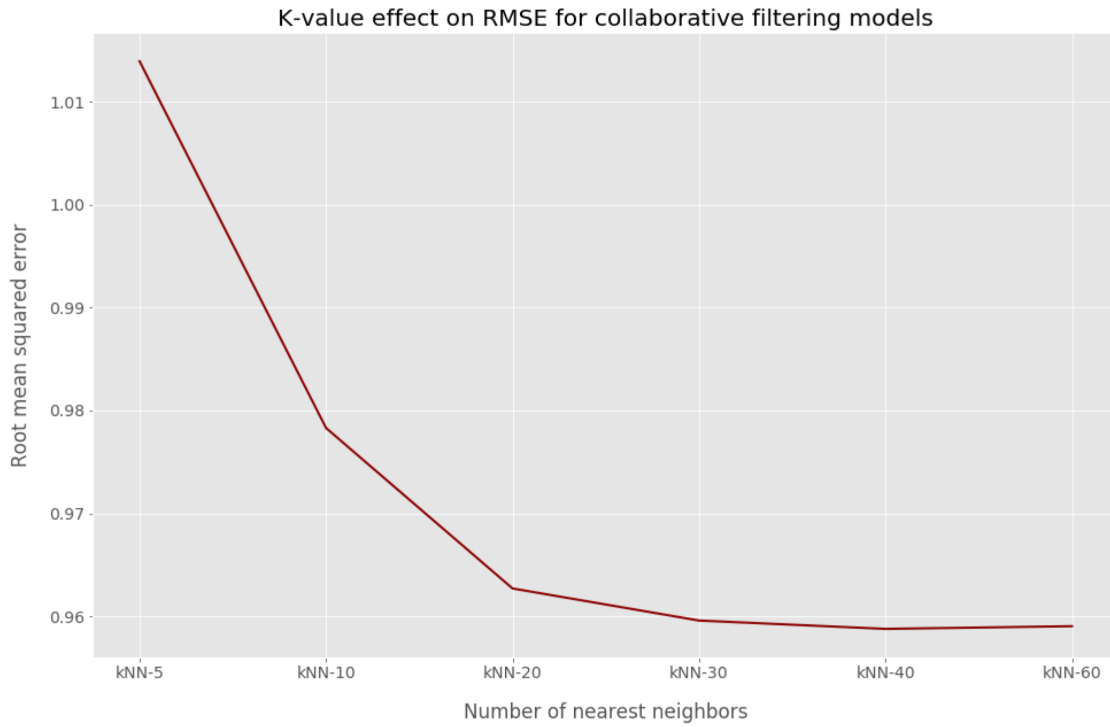
[28]:
```python
ml_algorithms = {'kNN-5': 5, 'kNN-10': 10, 'kNN-20': 20, 'kNN-30': 30, 'kNN-40':
 ↪ 40, "kNN-60": 60}

models_CF = []
RMSE_CF = []

for name, num_neighbours in ml_algorithms.items():
    predictions = []


    for index, row in X_val.iterrows():

        if row["FilmID"] in X_train["FilmID"].unique():

            users_rated_movie = X_train.loc[X_train['FilmID'] == row['FilmID'],␣
 ↪'BrukerID']

            users_sorted = (user_dist_matrix.loc[row['BrukerID'],␣
 ↪users_rated_movie].sort_values())

            nearest_neighbours = users_sorted[:num_neighbours]

            nn_data = train_df.loc[train_df['BrukerID'].isin(nearest_neighbours.
 ↪index.to_list())]
```

```
            nearest_neighbours_avg_rating = np.average(nn_data.
↪loc[train_df['FilmID'] == row['FilmID'], 'Rangering'],
                                                      axis=0, weights=(1/
↪nearest_neighbours))
        else:

            nearest_neighbours_avg_rating = 4

        if not np.isnan(nearest_neighbours_avg_rating):
            predictions.append(nearest_neighbours_avg_rating)
        else:
            predictions.append(3)

    models_CF.append(name)
    RMSE_CF.append(sqrt(mean_squared_error(y_val, predictions)))


RMSE_CF_dict = {"Model": models_CF, "RMSE": RMSE_CF}
RMSE_CF_df = pd.DataFrame(RMSE_CF_dict)
RMSE_CF_df
```

[28]:

	Model	RMSE
0	kNN-5	1.013958
1	kNN-10	0.978301
2	kNN-20	0.962699
3	kNN-30	0.959578
4	kNN-40	0.958769
5	kNN-60	0.959030

[41]:
```
fig7, ax7 = plt.subplots()
ax7.plot(RMSE_CF_df.Model, RMSE_CF_df.RMSE, label="RMSE", color='darkred',␣
↪linewidth=2)
plt.xlabel("Number of nearest neighbors", labelpad=18)
plt.ylabel("Root mean squared error", labelpad=15)
plt.title("K-value effect on RMSE for collaborative filtering models")
fig7.set_figheight(10)
fig7.set_figwidth(16)
plt.show()
```



[42]:
```
best_CF_model = []
RMSE_best_CF = []

CF_predictions = []

#
for index, row in X_val.iterrows():

    if row["FilmID"] in X_train["FilmID"].unique():

        users_rated_movie = X_train.loc[X_train['FilmID'] == row['FilmID'],␣
↪'BrukerID']

        users_sorted = (user_dist_matrix.loc[row['BrukerID'],␣
↪users_rated_movie].sort_values())

        nearest_neighbours = users_sorted[:40]

        nn_data = train_df.loc[train_df['BrukerID'].isin(nearest_neighbours.
↪index.to_list())]

        nearest_neighbours_avg_rating = np.average(nn_data.
↪loc[train_df['FilmID'] == row['FilmID'], 'Rangering'],
```

```
                                                    axis=0, weights=(1/
↪nearest_neighbours))
    else:

        nearest_neighbours_avg_rating = 4


    if not np.isnan(nearest_neighbours_avg_rating):
        CF_predictions.append(nearest_neighbours_avg_rating)
    else:
        CF_predictions.append(4)
```

[43]:

```
CBF_predictions = []
for index, row in X_val.iterrows():
    user_predictions = CBF_model.loc[row["BrukerID"], row["FilmID"]]
    CBF_predictions.append(user_predictions)


print("RMSE combined approach (Lasso and KNN-40):")
weighted_avgs = [(0.5, 0.5), (0.45, 0.55), (0.4, 0.6), (0.35, 0.65), (0.3, 0.
↪7), (0.25, 0.75), (0.20, 0.80)]

for weight in weighted_avgs:
    combined_predictions = np.array([y_pred * weight[0] for y_pred in np.
↪array(CBF_predictions)]) + np.array([y_pred * weight[1] for y_pred in np.
↪array(CF_predictions)])
    print(f"RMSE for combined approach with CBF weighted {weight[0]} and CF␣
↪weighted {weight[1]}: \n",
          sqrt(mean_squared_error(y_val, combined_predictions)), "\n")
```

```
RMSE combined approach (Lasso and KNN-40):
RMSE for combined approach with CBF weighted 0.5 and CF weighted 0.5:
 0.9383490270968214

RMSE for combined approach with CBF weighted 0.45 and CF weighted 0.55:
 0.9347989553339932

RMSE for combined approach with CBF weighted 0.4 and CF weighted 0.6:
 0.9324893533664824

RMSE for combined approach with CBF weighted 0.35 and CF weighted 0.65:
 0.9314294489645997

RMSE for combined approach with CBF weighted 0.3 and CF weighted 0.7:
 0.9316235074661314

RMSE for combined approach with CBF weighted 0.25 and CF weighted 0.75:
```

```
0.9330707464800498

RMSE for combined approach with CBF weighted 0.2 and CF weighted 0.8:
 0.9357653515804694
```

[44]:
```python
ratings_df = pd.read_csv("cleaned_data/rangering.csv")
movies_df = pd.read_csv("cleaned_data/film.csv")
users_df = pd.read_csv("cleaned_data/bruker.csv")
```

[45]:
```python
for index, row in movies_df.iterrows():
    title = row[1]
    release_year = int(title[-5:-1])
    movies_df.loc[index, 'Utgivelsesår'] = release_year

movies_df.head()
```

[45]:
```
   FilmID                                    Tittel  Action  Adventure  \
0       0                  Autumn in New York (2000)       0          0
1       1  Vie est belle, La (Life is Rosey) (1987)       0          0
2       2                     Defying Gravity (1997)       0          0
3       3                     Ruthless People (1986)       0          0
4       4                   Portraits Chinois (1996)       0          0

   Animation  Children's  Comedy  Crime  Documentary  Drama  …  Film-Noir  \
0          0           0       0      0            0      1  …          0
1          0           0       1      0            0      1  …          0
2          0           0       0      0            0      1  …          0
3          0           0       1      0            0      0  …          0
4          0           0       0      0            0      1  …          0

   Horror  Musical  Mystery  Romance  Sci-Fi  Thriller  War  Western  \
0       0        0        0        1       0         0    0        0
1       0        0        0        0       0         0    0        0
2       0        0        0        0       0         0    0        0
3       0        0        0        0       0         0    0        0
4       0        0        0        0       0         0    0        0

   Utgivelsesår
0        2000.0
1        1987.0
2        1997.0
3        1986.0
4        1996.0

[5 rows x 21 columns]
```

[63]:
```python
ratings_users_merged = users_df.merge(ratings_df, on=['BrukerID'])
movie_fans = {movie_id: [] for movie_id in movies_df['FilmID']}

for index, row in ratings_users_merged.iterrows():
    rating = row['Rangering']
    movie_id = row['FilmID']
    user_age = row['Alder']

    if int(rating) > 3:
        movie_fans[movie_id].append(user_age)


fan_avg_ages = []

for movie_id, ages in movie_fans.items():
    if len(ages) > 0:
        fan_avg_ages.append(np.mean(ages))
    else:
        fan_avg_ages.append(np.NaN)

movies_df['FAA'] = fan_avg_ages

print(f'There are {movies_df.isnull().sum().sum()} missing values for "FAA"')

movies_df['FAA'].fillna(29.894868, inplace=True)
movies_df.head()
```

```
There are 364 missing values for "FAA"
```

[63]:
```
   FilmID                                  Tittel  Action  Adventure  \
0       0                 Autumn in New York (2000)       0          0
1       1  Vie est belle, La (Life is Rosey) (1987)       0          0
2       2                   Defying Gravity (1997)       0          0
3       3                   Ruthless People (1986)       0          0
4       4                  Portraits Chinois (1996)       0          0

   Animation  Children's  Comedy  Crime  Documentary  Drama  …  Horror  \
0          0           0       0      0            0      1  …       0
1          0           0       1      0            0      1  …       0
2          0           0       0      0            0      1  …       0
3          0           0       1      0            0      0  …       0
4          0           0       0      0            0      1  …       0

   Musical  Mystery  Romance  Sci-Fi  Thriller  War  Western  Utgivelsesår  \
0        0        0        1       0         0    0        0        2000.0
1        0        0        0       0         0    0        0        1987.0
2        0        0        0       0         0    0        0        1997.0
```

```
3           0           0           0          0            0     0           0           1986.0
4           0           0           0          0            0     0           0           1996.0

          FAA
0   30.416667
1   29.833333
2   30.000000
3   31.884259
4   29.894868

[5 rows x 22 columns]
```

[96]:
```
movies_df.to_csv('movies_feature_engineered.csv', index=False)
```

[64]:
```
X = ratings_df.drop(columns='Rangering')
y = ratings_df["Rangering"].values

X_train, X_val_and_test, y_train, y_val_and_test = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=101)
X_val, X_test, y_val, y_test = train_test_split(X_val_and_test, y_val_and_test,␣
 ↪test_size=0.5, random_state=101)

train_df = X_train.copy()
train_df["Rangering"] = y_train

train_df
```

[64]:

	BrukerID	FilmID	Tidstempel	Rangering
708938	4341	1210	976573899	3
371257	1425	3192	970774826	4
845603	5565	2299	976239652	5
667558	3970	83	965770467	3
821518	5320	3843	976249576	4
…	…	…	…	…
661055	3915	1846	974698675	4
204614	32	884	974679433	4
476497	2357	101	975126955	4
214539	122	2098	974662091	4
176991	5383	2759	962992451	5

[630081 rows x 4 columns]

[65]:
```
content_train_df = pd.merge(train_df, movies_df, on='FilmID')
content_train_df.drop(columns=['Tidstempel', 'FilmID', 'Tittel'], inplace=True)

y_grouped_by_user = content_train_df.groupby(["BrukerID"])
y_train_listed = []
```

```
for i, j in y_grouped_by_user:
    y_train_listed.append(j["Rangering"].values)


y_train_listed[0]

content_train_df.drop(columns='Rangering', inplace=True)
x_grouped_by_user = content_train_df.groupby(["BrukerID"])
x_train_listed = []

for user_id, group in x_grouped_by_user:
    x_train_listed.append(group.drop(columns='BrukerID'))


all_movies = movies_df.drop(columns=['Tittel', 'FilmID'])
all_movies


user_ids = []
for user_id, group in x_grouped_by_user:
    user_ids.append(user_id)

movie_ids = movies_df["FilmID"].values

df_val = X_val.copy()
df_val["Rangering"] = y_val
validation_matrix = pd.DataFrame(index=user_ids, columns=movie_ids)
for array in df_val.to_records():
    user = array['BrukerID']
    movie = array['FilmID']
    true_rating = array['Rangering']
    validation_matrix.loc[user][movie] = true_rating
```

[68]:
```
ml_algorithms = {"Lasso": Lasso(alpha=1.0, max_iter=10000), "KNN_7":␣
↪KNeighborsRegressor(n_neighbors=7),
                 "SVR": SVR(C=1.0)}

improved_models_listed = []
improved_models_RMSE = []

for name, ml_alg in ml_algorithms.items():

    CBF_predictions = []


    for i, x in enumerate(x_train_listed):
```

```
        ml_alg.fit(x_train_listed[i], y_train_listed[i])

        prediction = ml_alg.predict(all_movies)
        prediction = np.clip(prediction, 1, 5)

        CBF_predictions.append(prediction)


    CBF_y_pred_df = pd.DataFrame(CBF_predictions, index=user_ids,
 ↪columns=movie_ids)


    num_actual = validation_matrix.to_numpy().flatten()[validation_matrix.
 ↪notna().to_numpy().flatten()]
    num_predict = CBF_y_pred_df.to_numpy().flatten()[validation_matrix.notna().
 ↪to_numpy().flatten()]

    improved_models_RMSE.append(sqrt(mean_squared_error(num_predict,
 ↪num_actual)))
    improved_models_listed.append(name)


RMSE_content_df_improved = pd.DataFrame({"Model": improved_models_listed,
 ↪"RMSE": improved_models_RMSE})
print("RMSE of different content-based filtering models, including the year of
 ↪release feature")
RMSE_content_df_improved
```

```
C:\Users\sebas\anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.0, tolerance: 0.0
  positive)
```

```
RMSE of different content-based filtering models, including the year of release
feature
```

[68]:

	Model	RMSE
0	Lasso	1.020758
1	KNN_7	1.040054
2	SVR	1.070242

[72]:

```
model = Lasso(alpha=1.0, max_iter=10000)
CBF_improved_predictions = []

for i, j in enumerate(x_train_listed):
```

```
    model.fit(x_train_listed[i], y_train_listed[i])
    prediction = model.predict(all_movies)
    prediction = np.clip(prediction, 1, 5)
    CBF_improved_predictions.append(prediction)

CBF_improved_model = pd.DataFrame(CBF_improved_predictions, index=user_ids,␣
 ↪columns=movie_ids)

num_actual = validation_matrix.to_numpy().flatten()[validation_matrix.notna().
 ↪to_numpy().flatten()]
num_predict = CBF_improved_model.to_numpy().flatten()[validation_matrix.notna().
 ↪to_numpy().flatten()]
print("RMSE of best content-based filtering model:",␣
 ↪sqrt(mean_squared_error(num_predict, num_actual)))

CBF_improved_model.to_pickle("./CBF_model.pkl")
```

```
C:\Users\sebas\anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.0, tolerance: 0.0
  positive)
```

```
RMSE of best content-based filtering model: 1.0207579360879961
```

[77]:
```
CBF_y_pred = []
for index, row in X_val.iterrows():
    user_predictions = CBF_improved_model.loc[row["BrukerID"], row["FilmID"]]
    CBF_y_pred.append(user_predictions)

print("RMSE combined approach (Lasso and KNN-40):")
weighted_avgs = [(0.5, 0.5), (0.45, 0.55), (0.4, 0.6), (0.35, 0.65), (0.3, 0.
 ↪7), (0.25, 0.75), (0.20, 0.80)]
for weight in weighted_avgs:
    combined_predictions = ((np.array(CBF_y_pred) * weight[0]) + (np.
 ↪array(CF_predictions)) * weight[1])
    print(f"RMSE for combined approach with CBF weighted {weight[0]} and CF␣
 ↪weighted {weight[1]}: \n",
          sqrt(mean_squared_error(y_val, combined_predictions)), "\n")
```

```
RMSE combined approach (Lasso and KNN-40):
RMSE for combined approach with CBF weighted 0.5 and CF weighted 0.5:
 0.9310363505982111

RMSE for combined approach with CBF weighted 0.45 and CF weighted 0.55:
 0.9283464888895532
```

```
RMSE for combined approach with CBF weighted 0.4 and CF weighted 0.6:
 0.9268766505281787

RMSE for combined approach with CBF weighted 0.35 and CF weighted 0.65:
 0.926632641189354

RMSE for combined approach with CBF weighted 0.3 and CF weighted 0.7:
 0.9276154282369545

RMSE for combined approach with CBF weighted 0.25 and CF weighted 0.75:
 0.9298211216418713

RMSE for combined approach with CBF weighted 0.2 and CF weighted 0.8:
 0.9332410505126286
```

[82]:

```
CF_predictions_test = []
for index, row in X_test.iterrows():
    if row["FilmID"] in X_train["FilmID"].unique():
        users_rated_movie = X_train.loc[X_train['FilmID'] == row['FilmID'],␣
 ↪'BrukerID']
        users_sorted = (user_dist_matrix.loc[row['BrukerID'],␣
 ↪users_rated_movie].sort_values())
        n_neighbours = users_sorted[:40]
        nn_data = train_df.loc[train_df['BrukerID'].isin(n_neighbours.index.
 ↪to_list())]
        nearest_neighbours_avg_rating = np.average(nn_data.
 ↪loc[train_df['FilmID'] == row['FilmID'], 'Rangering'],
                                                   axis=0, weights=(1/
 ↪n_neighbours))
    else:
        nearest_neighbours_avg_rating = train_df["Rangering"].mean()

    if not np.isnan(nearest_neighbours_avg_rating):
        CF_predictions_test.append(nearest_neighbours_avg_rating)
    else:
        CF_predictions_test.append(4)

print("RMSE KNN_40:", sqrt(mean_squared_error(y_test, CF_predictions_test)))
```

```
RMSE KNN_40: 0.958015760014762
```

[92]:

```
CBF_predictions_test = []
for index, row in X_test.iterrows():
    user_predictions = CBF_improved_model.loc[row["BrukerID"], row["FilmID"]]
    CBF_predictions_test.append(user_predictions)
```

```python
print("RMSE Lasso:", sqrt(mean_squared_error(y_test, CBF_predictions_test)))
```

RMSE Lasso: 1.019278388038111

[95]:
```python
hybrid_predictions_test = (np.array([y_pred * 0.35 for y_pred in np.
↪array(CBF_predictions_test)])
                           + np.array([y_pred * 0.65 for y_pred in np.
↪array(CF_predictions_test)]))

print(f"RMSE hybrid recommendations (test data):␣
↪{sqrt(mean_squared_error(y_test, hybrid_predictions_test))} ")
```

RMSE hybrid recommendations (test data): 0.9254351615504374