20220524_Wirtschaftsinformatik_FAT2

May 24, 2022

[1]:
```
"""Filmempfehlungssystem"""
```

[1]:
```
'Filmempfehlungssystem'
```

[2]:
```
"""Kundenempfehlungssystem oder Produktempfehlungssystem,..."""
```

[2]:
```
'Kundenempfehlungssystem oder Produktempfehlungssystem,…'
```

[3]:
```
"""
1. Nutzerbewertungen
2. Nutzerdaten
"""
```

[3]:
```
'1. Nutzerbewertungen 2. Nutzerdaten'
```

[4]:
```
"""
Die Idee hinter dem inhaltsbasierten (kognitiven Filter) Empfehlungssystem ist␣
 ↪es,
einen Artikel auf der Grundlage eines Vergleichs zwischen dem Inhalt des␣
 ↪Artikels
und einem Benutzerprofil zu empfehlen.
"""
```

[4]:
```
'Die Idee hinter dem inhaltsbasierten (kognitiven Filter) Empfehlungssystem ist
es, \neinen Artikel auf der Grundlage eines Vergleichs zwischen dem Inhalt des
Artikels \nund einem Benutzerprofil zu empfehlen.'
```

[5]:
```
"""K Means Clustering. Nearest Neighbour"""
```

[5]:
```
'K Means Clustering. Nearest Neighbour'
```

0.1 Daten Vorbereitung

[6]:
```
# importing basic libraries
import pandas as pd
import numpy as np
```

[7]:
```
# usecols allows us to select our choice of features
movies_df=pd.read_csv('/Users/h4/desktop/movies.csv',
                      usecols=['movieId','title'], dtype={'movieId':
 ↪'int32','title':'str'})
movies_df.head()
```

[7]:

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)

[8]:
```
ratings_df=pd.read_csv('/Users/h4/desktop/ratings.csv',
    usecols=['userId', 'movieId', 'rating','timestamp'],dtype={'userId':␣
 ↪'int32', 'movieId': 'int32', 'rating': 'float32'})
ratings_df.head()
```

[8]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

[10]:
```
# checkinng for NaN values

movies_df.isnull().sum()
```

[10]:
```
movieId    0
title      0
dtype: int64
```

[11]:
```
ratings_df.isnull().sum()
```

[11]:
```
userId       0
movieId      0
rating       0
timestamp    0
dtype: int64
```

[12]:
```
# Form (Shape) der Daten

print("Movies:",movies_df.shape)
print("Ratings:",ratings_df.shape)
```

```
Movies: (9742, 2)
Ratings: (100836, 4)
```

[13]:
```
# Prüfungsrelevant (Zusammenbringen von zwei Dataframes)

# merging for analysis

#movies_df.info()
#ratings_df.info()

movies_merged_df=movies_df.merge(ratings_df, on='movieId') # movieId, weil wir␣
↪eine Movie-Empfehlung abgeben wollen.
movies_merged_df.head()
```

[13]:

	movieId	title	userId	rating	timestamp
0	1	Toy Story (1995)	1	4.0	964982703
1	1	Toy Story (1995)	5	4.0	847434962
2	1	Toy Story (1995)	7	4.5	1106635946
3	1	Toy Story (1995)	15	2.5	1510577970
4	1	Toy Story (1995)	17	4.5	1305696483

[14]:
```
# Prüfungsrelevant

# löschung von NaN Daten

movies_merged_df=movies_merged_df.dropna(axis = 0, subset = ['title'])
movies_merged_df.head()
```

[14]:

	movieId	title	userId	rating	timestamp
0	1	Toy Story (1995)	1	4.0	964982703
1	1	Toy Story (1995)	5	4.0	847434962
2	1	Toy Story (1995)	7	4.5	1106635946
3	1	Toy Story (1995)	15	2.5	1510577970
4	1	Toy Story (1995)	17	4.5	1305696483

[16]:
```
movies_average_rating=movies_merged_df.groupby('title')['rating'].mean().
↪sort_values(ascending=False).reset_index().rename(columns={'rating':'Average␣
↪Rating'})
movies_average_rating.head()
```

[16]:

	title	Average Rating
0	Gena the Crocodile (1969)	5.0

```
1                  True Stories (1986)             5.0
2        Cosmic Scrat-tastrophe (2015)             5.0
3              Love and Pigeons (1985)             5.0
4  Red Sorghum (Hong gao liang) (1987)             5.0
```

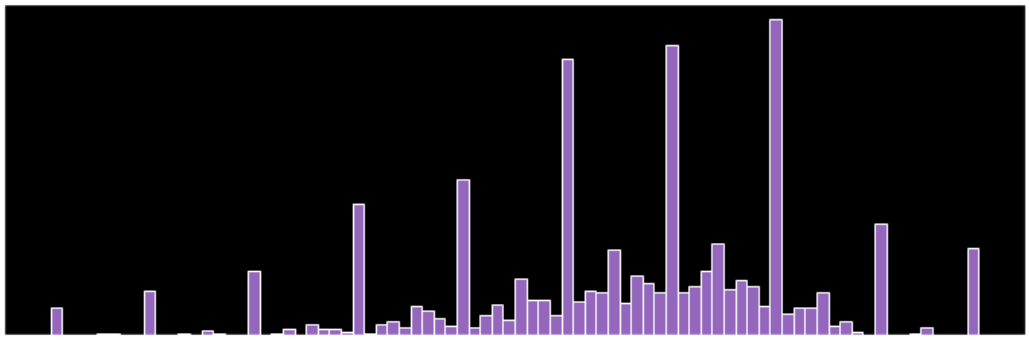
[17]:
```
movies_rating_count=movies_merged_df.groupby('title')['rating'].count().
↪sort_values(ascending=True).reset_index().rename(columns={'rating':'Rating␣
↪Count'}) #ascending=False
movies_rating_count_avg=movies_rating_count.merge(movies_average_rating,␣
↪on='title')
movies_rating_count_avg.head()
```

[17]:
```
                                               title  Rating Count  \
0                                       '71 (2014)              1
1                                   Latter Days (2003)             1
2                               Late Shift, The (1996)             1
3  Late Night with Conan O'Brien: The Best of Tri…              1
4                           Late Night Shopping (2001)             1

   Average Rating
0             4.0
1             3.5
2             2.5
3             2.0
4             4.5
```

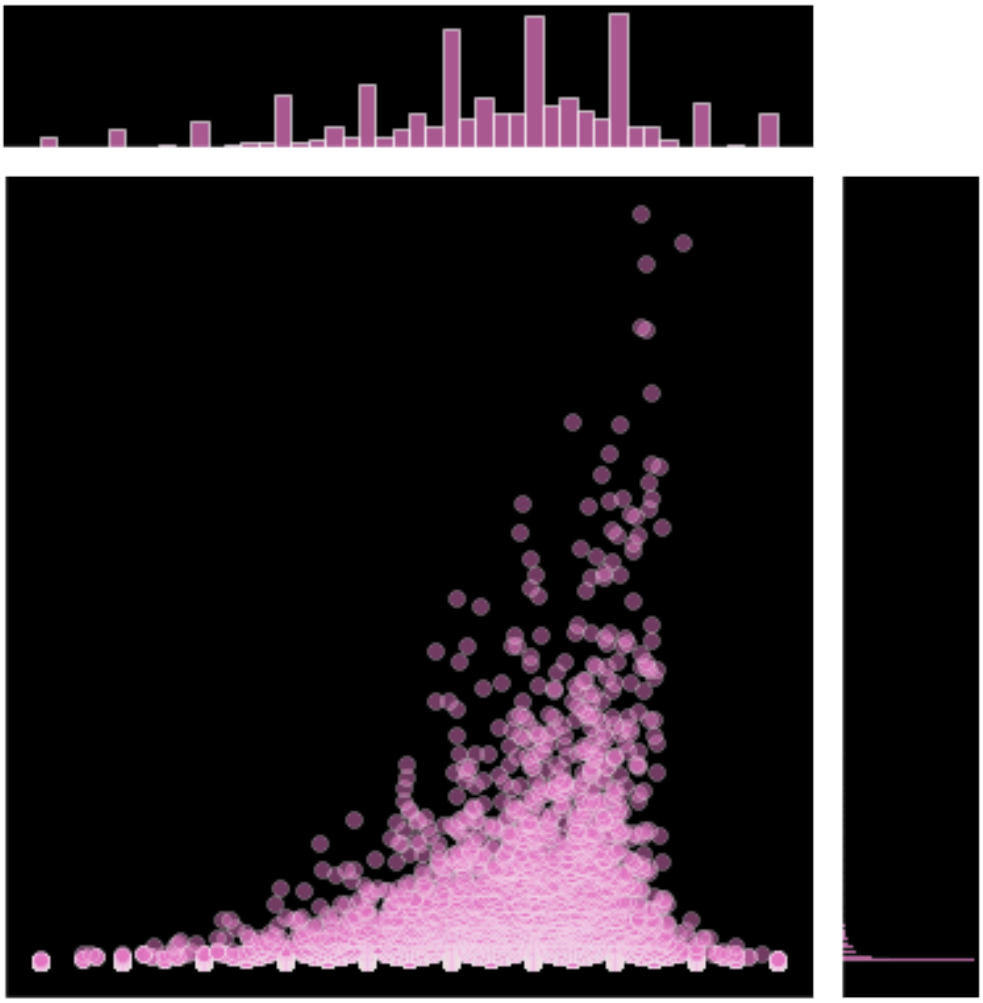
[18]:
```
# Prüfungsrelevant

# Data Visualization

#importing visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(font_scale = 1)
plt.rcParams["axes.grid"] = False
plt.style.use('dark_background')
%matplotlib inline
```

[20]:
```
# Prüfungsrelevant

plt.figure(figsize=(12,4))
plt.hist(movies_rating_count_avg['Rating Count'],bins=80,color='tab:purple')
plt.ylabel('Ratings Count(Scaled)', fontsize=16)
plt.savefig('/Users/h4/desktop/ratingcounthist.jpg')
```

[22]:
```
# Prüfungsrelevant

plt.figure(figsize=(12,4))
plt.hist(movies_rating_count_avg['Average Rating'],bins=80,color='tab:purple')
plt.ylabel('Average Rating',fontsize=16)
plt.savefig('/Users/h4/desktop/avgratinghist.jpg')
```



[24]:
```
# Prüfungsrelevant (Darstellung, welche BEIDE Hisograme zusammen darstellt!)

plot=sns.jointplot(x='Average Rating',y='Rating␣
 ↪Count',data=movies_rating_count_avg,alpha=0.5, color='tab:pink')
plot.savefig('/Users/h4/desktop/joinplot.jpg')
```



[25]:
```
# Prüfungsrelevant

"""Analyse -

Diagramm Nr. 1 bestätigt unsere Beobachtungen einer hohen Anzahl von Filmen
mit niedriger Bewertungszahl.

Histogramm Nr. 2 zeigt die Verteilungsfunktion der durchschnittlichen␣
 ↪Bewertungswerte.

Der Joinplot veranschaulicht sehr schön,
dass es nur eine Teilmenge von Werten mit höheren Bewertungen gibt,
die eine beträchtliche Anzahl von Bewertungen aufweisen.
```

```
"""
```

[25]:
```
'Analyse -\n\nDiagramm Nr. 1 bestätigt unsere Beobachtungen einer hohen Anzahl
von Filmen \nmit niedriger Bewertungszahl.\n\nHistogramm Nr. 2 zeigt die
Verteilungsfunktion der durchschnittlichen Bewertungswerte.\n\nDer Joinplot
veranschaulicht sehr schön, \ndass es nur eine Teilmenge von Werten mit höheren
Bewertungen gibt, \ndie eine beträchtliche Anzahl von Bewertungen aufweisen.\n'
```

[27]:
```
# Outliers! (Punkte die sich nicht "normal verhalten" und somit die␣
 ↪statistische Bewertung stören!)
# Outliers sind idR zu entfernen! (immer)

rating_with_RatingCount = movies_merged_df.merge(movies_rating_count, left_on =␣
 ↪'title', right_on = 'title', how = 'left')
rating_with_RatingCount.head()
```

[27]:

	movieId	title	userId	rating	timestamp	Rating Count
0	1	Toy Story (1995)	1	4.0	964982703	215
1	1	Toy Story (1995)	5	4.0	847434962	215
2	1	Toy Story (1995)	7	4.5	1106635946	215
3	1	Toy Story (1995)	15	2.5	1510577970	215
4	1	Toy Story (1995)	17	4.5	1305696483	215

[28]:
```
pd.set_option('display.float_format', lambda x: '%.3f' % x)
print(rating_with_RatingCount['Rating Count'].describe())
```

```
count   100836.000
mean        58.759
std         61.965
min          1.000
25%         13.000
50%         39.000
75%         84.000
max        329.000
Name: Rating Count, dtype: float64
```

[29]:
```
# ein Threshold ist ein Wert, unter dem die Daten gelöscht werden

popularity_threshold = 50
popular_movies= rating_with_RatingCount[rating_with_RatingCount['Rating␣
 ↪Count']>=popularity_threshold]
popular_movies.head()
#popular_movies.shape
```

[29]:

	movieId	title	userId	rating	timestamp	Rating Count
0	1	Toy Story (1995)	1	4.000	964982703	215
1	1	Toy Story (1995)	5	4.000	847434962	215

```
2        1  Toy Story (1995)       7   4.500  1106635946           215
3        1  Toy Story (1995)      15   2.500  1510577970           215
4        1  Toy Story (1995)      17   4.500  1305696483           215
```

[31]:
```
# Erzeugung einer Pivottabelle mit den UserIds und die Movie Bewertungen

import os
movie_features_df=popular_movies.
 ↪pivot_table(index='title',columns='userId',values='rating').fillna(0)
movie_features_df.head()
```

[31]:
```
userId                              1     2     3     4     5     6     7    \
title
10 Things I Hate About You (1999) 0.000 0.000 0.000 0.000 0.000 0.000 0.000
12 Angry Men (1957)               0.000 0.000 0.000 5.000 0.000 0.000 0.000
2001: A Space Odyssey (1968)      0.000 0.000 0.000 0.000 0.000 0.000 4.000
28 Days Later (2002)              0.000 0.000 0.000 0.000 0.000 0.000 0.000
300 (2007)                        0.000 0.000 0.000 0.000 0.000 0.000 0.000

userId                              8     9     10   …   601   602   603  \
title                                                …
10 Things I Hate About You (1999) 0.000 0.000 0.000  … 0.000 0.000 3.000
12 Angry Men (1957)               0.000 0.000 0.000  … 5.000 0.000 0.000
2001: A Space Odyssey (1968)      0.000 0.000 0.000  … 0.000 0.000 5.000
28 Days Later (2002)              0.000 0.000 0.000  … 0.000 0.000 0.000
300 (2007)                        0.000 0.000 3.000  … 0.000 0.000 0.000

userId                              604   605   606   607   608   609   610
title
10 Things I Hate About You (1999) 0.000 5.000 0.000 0.000 0.000 0.000 0.000
12 Angry Men (1957)               0.000 0.000 0.000 0.000 0.000 0.000 0.000
2001: A Space Odyssey (1968)      0.000 0.000 5.000 0.000 3.000 0.000 4.500
28 Days Later (2002)              0.000 0.000 0.000 0.000 3.500 0.000 5.000
300 (2007)                        0.000 3.000 0.000 0.000 5.000 0.000 4.000

[5 rows x 606 columns]
```

0.2 Filmempfehlungsmodel (Nearest Neighbours)

[32]:
```
# Schaffen wir eine Matrix (Sparse Matrix aus den Daten)

from scipy.sparse import csr_matrix
movie_features_df_matrix = csr_matrix(movie_features_df.values)
```

[33]:
```
# Prüfungsrelevant

from sklearn.neighbors import NearestNeighbors
model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
model_knn.fit(movie_features_df_matrix)
```

[33]:
```
NearestNeighbors(algorithm='brute', metric='cosine')
```

[34]:
```
movie_features_df.shape
```

[34]:
```
(450, 606)
```

[44]:
```
# Prüfungsrelevant

# wir nehmen einen "Random" Movie und bewerten den Abstand zu den anderen␣
 ↪Filmen.
# wir nehmen dann die Movies die einen höheren Abstand zu der Wahl haben.

query_index = np.random.choice(movie_features_df.shape[0])
print(query_index)
distances, indices = model_knn.kneighbors(movie_features_df.iloc[query_index,:].
 ↪values.reshape(1, -1), n_neighbors = 6)
```

```
208
```

[46]:
```
# Prüfungsrelevant

for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {0}:\n'.format(movie_features_df.
 ↪index[query_index]))
    else:
        print('{0}: {1}, with distance of {2}:'.format(i, movie_features_df.
 ↪index[indices.flatten()[i]], distances.flatten()[i]))
```

```
Recommendations for I Am Legend (2007):

1: I, Robot (2004), with distance of 0.4059225916862488:
2: Day After Tomorrow, The (2004), with distance of 0.4223892092704773:
3: Avatar (2009), with distance of 0.4473915696144104:
4: Hangover, The (2009), with distance of 0.46926259994506836:
5: Star Wars: Episode III - Revenge of the Sith (2005), with distance of
0.4880126118659973:
```

[ ]:
```

```