Sprawozdanie z realizacji projektu "Analizy filmów na podstawie https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset" wykonywanego w ramach przedmiotu Podstawy Reprezentacji i Analizy Danych.

Celem projektu było stworzenie modelu mogącego dopasować filmy do podanych gatunków na podstawie dostępnych ze zbioru danych.

Członkowie zespołu: Jordan Parviainen, Jakub Niewadzi, Szymon Posiadała.

W celu zrealizowania projektu wykorzystaliśmy uczenie nadzorowane, a konkretnie naiwny algorytm Bayesa oraz SVC. Użyliśmy dwóch algorytmów, aby można było porównać wyniki uczeia się dla obu i wybrać lepszy.

Pobrane dane mają strukturę tablicy o kolumnach id, genres, overview, id, release_date, vote_count, vote_average, runtime, title. Pobraliśmy 45 466 takich kolumn, natomiast po odrzuceniu skrajnych plików pozostało nam 31524 Na tym etapie byliśmy w stanie wyznaczyć podstawowe wartości, takie jak średnia ocen wszystkich filmów, średnia długość czasu trwania, oraz zobaczyć histogramy interesujących nas wartości

Importowanie bibliotek niezbędnych do działania programu:

```
from azureml.core import Workspace, Dataset
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn import tree
from sklearn.model_selection import train_test_split
from skimage import io
from skimage.color import rgb2gray
from skimage.exposure import histogram
from datetime import datetime
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from nltk.stem import WordNetLemmatizer
from collections import OrderedDict
from sklearn.feature_extraction.text import CountVectorizer
from collections import OrderedDict
import string
from sklearn.ensemble import VotingClassifier
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
pd.options.display.float_format = "{:.3f}".format
```

Konfiguracja Azure i pobranie danych:

```
subscription_id = '2e950711-3856-43c9-a2db-c2c17d51a26e'
resource_group = 'PRiAD'
workspace_name = 'PRIADfilmy'
workspace = Workspace(subscription_id, resource_group, workspace_name)

dataset = Dataset.get_by_name(workspace,
name='Movies_data_preprocessed')
dataset.download(target_path='.', overwrite=True)
```

```
['/mnt/batch/tasks/shared/LS_root/mounts/clusters/priadkuba/code/
Users/01169610/_meta.yaml',

'/mnt/batch/tasks/shared/LS_root/mounts/clusters/priadkuba/code/Users/
01169610/data.csv',

'/mnt/batch/tasks/shared/LS_root/mounts/clusters/priadkuba/code/Users/
01169610/schema/_schema.json',

'/mnt/batch/tasks/shared/LS_root/mounts/clusters/priadkuba/code/Users/
01169610/_samples.json',

'/mnt/batch/tasks/shared/LS_root/mounts/clusters/priadkuba/code/Users/
01169610/data.visualization']
```

Analizowane dane pochodzą ze zbiorów Movielens, pobranych przy pomocy strony https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

Po wczytaniu początkowych danych do 'data.csv' otrzymaliśmy plik daleki od naszych oczekiwań. Przed obróbką 'dane.csv' zawierały między innymi:

- filmy o długości poniżej 1 minuty;
- filmy o długości powyżej 5 godzin, z najdłuższym trwającym około 1200 minut;
- filmy bez ocen;
- filmy bez opisów;
- bardzo stare filmy, sprzed 1950 roku;
- filmy nie przypisane do żadnego gatunku;
- gatunki filmów zbite w pseudo-json'y;
- filmy przynależące do wielu gatunków

Informacje te uzyskaliśmy między innymi analizując poniższe wykresy, jak również surowe pliki z danymi.

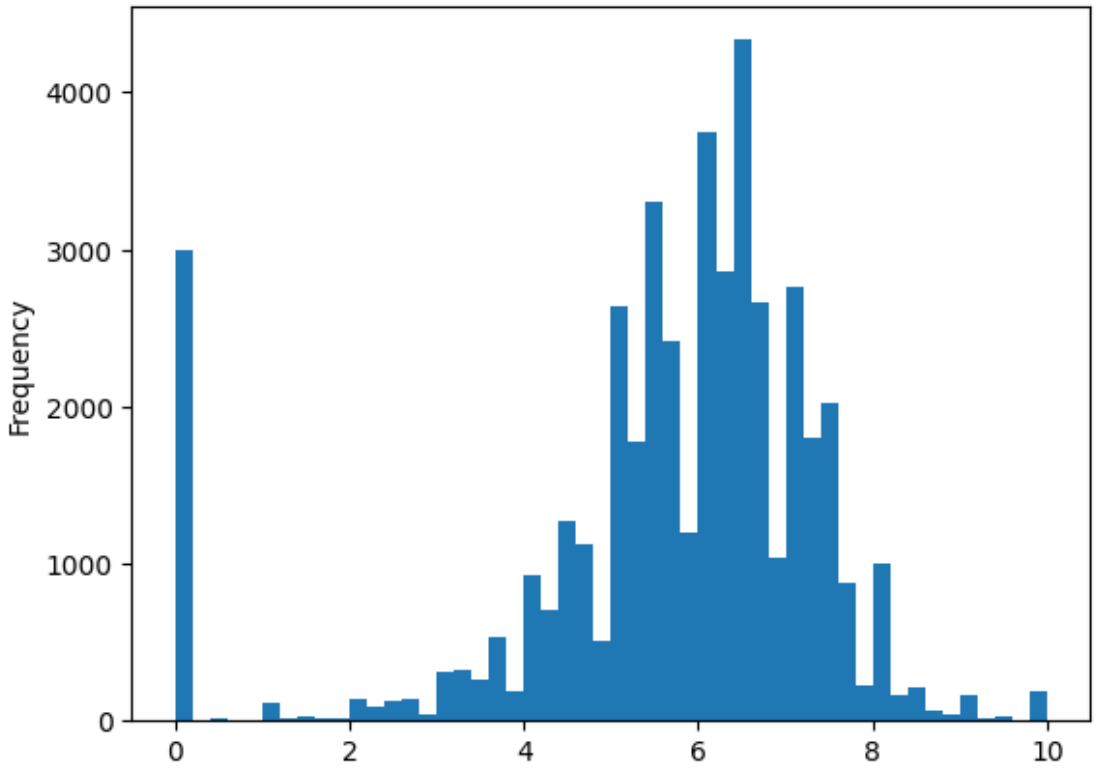
Pobieranie pliku i początki analizy eksploracyjnej:

```
df = pd.read_csv('data.csv')
df.info()

plt.figure(figsize=(10,10), dpi= 80)

#Wykres, średnia i odchylenie standardowe oceny filmów
plt.show()
df['vote_average'].plot.hist(bins=50)
print("Average rating mean:"+str(df['vote_average'].mean())+' standard
deviation:'+str(df['vote_average'].std()))
print("Average rating varies from "+str(df['vote_average'].min())+" to
"+str(df['vote_average'].max()))
plt.show()

#Wykres, średnia i odchylenie standardowe czasu trwania filmów
df['runtime'].plot.hist(bins=100)
print("Runtime mean:"+str(df['runtime'].mean())+' standard
deviation:'+str(df['runtime'].std()))
print("Runtime varies from "+str(df['runtime'].min())+" to
"+str(df['runtime'].max()))
plt.show()

#Wykres, średnia i odchylenie standardowe ilości recenzji
df['vote_count'].plot.hist(bins=100,log=True)
print("Vote count mean:"+str(df['vote_count'].mean())+' standard
deviation:'+str(df['vote_count'].std()))
print("Vote count varies from "+str(df['vote_count'].min())+" to
"+str(df['vote_count'].max()))
plt.show()
print()

#Ilość i rodzaje gatunków
print("Number of unique genres:"+str(df['genres'].nunique()))
print("Unique genres:"+df['genres'].unique())
```

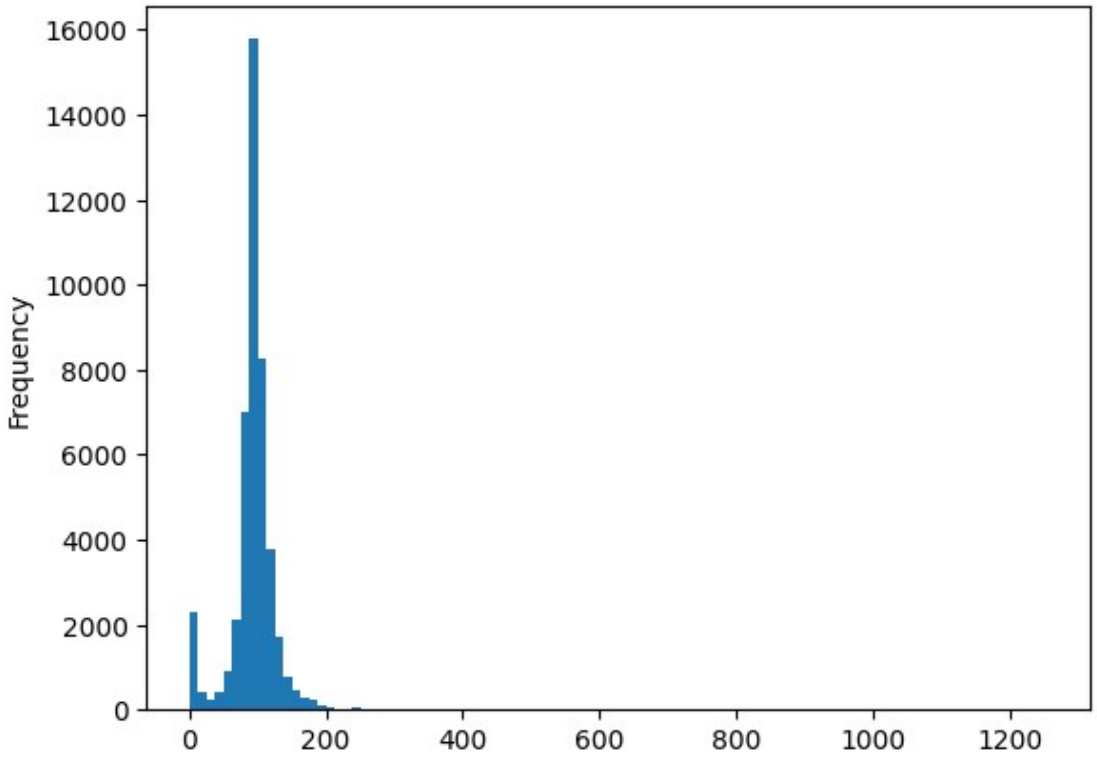
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   genres        45466 non-null  object
 1   id            45466 non-null  object
 2   overview      44507 non-null  object
 3   release_date  45379 non-null  object
 4   runtime       45203 non-null  float64
 5   title         45460 non-null  object
 6   vote_average  45460 non-null  float64
 7   vote_count    45460 non-null  float64
dtypes: float64(3), object(5)
memory usage: 2.8+ MB
```

<Figure size 800x800 with 0 Axes>

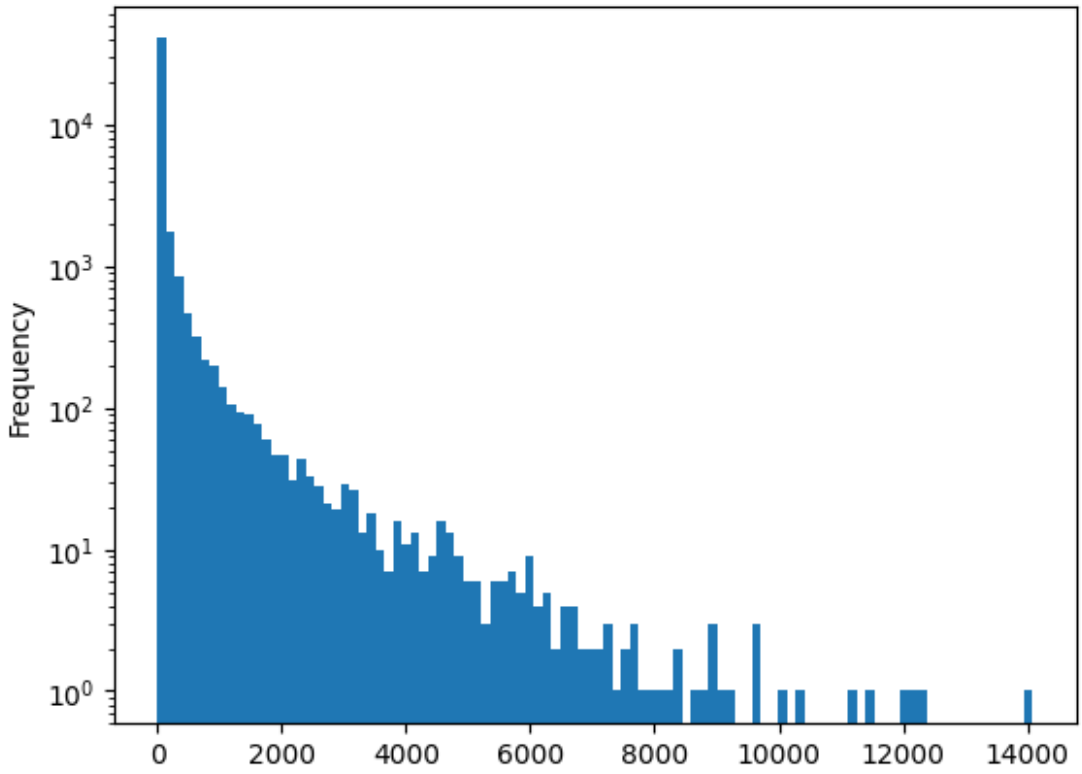
Average rating mean:5.618207215133889 standard deviation:1.9242159915229755
Average rating varies from 0.0 to 10.0



Runtime mean:94.12819945578833 standard deviation:38.40781048550736
Runtime varies from 0.0 to 1256.0



Vote count mean:109.89733831940167 standard
deviation:491.3103739397137
Vote count varies from 0.0 to 14075.0




```
Number of unique genres:4069
["Unique genres:[{'id': 16, 'name': 'Animation'}, {'id': 35, 'name':
'Comedy'}, {'id': 10751, 'name': 'Family'}]"
 "Unique genres:[{'id': 12, 'name': 'Adventure'}, {'id': 14, 'name':
'Fantasy'}, {'id': 10751, 'name': 'Family'}]"
 "Unique genres:[{'id': 10749, 'name': 'Romance'}, {'id': 35, 'name':
'Comedy'}]"
 ...
 "Unique genres:[{'id': 80, 'name': 'Crime'}, {'id': 35, 'name':
'Comedy'}, {'id': 28, 'name': 'Action'}, {'id': 10751, 'name':
'Family'}]"
 "Unique genres:[{'id': 28, 'name': 'Action'}, {'id': 9648, 'name':
'Mystery'}, {'id': 53, 'name': 'Thriller'}, {'id': 27, 'name':
'Horror'}]"
 "Unique genres:[{'id': 10751, 'name': 'Family'}, {'id': 16, 'name':
'Animation'}, {'id': 10749, 'name': 'Romance'}, {'id': 35, 'name':
'Comedy'}]"]
```

Pierwszy wykres przedstawia średnią ocenę filmów, dzięki niemu widzimy że dużo filmów miały ocenę 0. Drugi wykres przedstawia czas trwania filmów. Można na nim zauważyć bardzo małą liczbę filmów trwających dłużej niż 300 minut. Trzeci wykres przedstawia liczbę ocen filmów, widać na nim że spora ilość filmów nie ma ocen i mogą być zbędnymi danymi. Zbierając zdobyte infromacje z wykresów można było zauważyć, że jest nadmiarowa liczba filmów mająca zarówno oceny 0, jak i 0 oddanych głosów, co sugeruje

powiązanie między tymi parametrami. Po usunięciu brakujących danych, jak i "odchudzeniu" zbioru na podstawie podstawowych informacji otrzymaliśmy następujące dane.

```
#usunięcie brakujących danych
df = df.dropna()
print("liczba obiektów: ", df.count(0)[0])

df=df.sort_values('runtime', ascending=False)
#usunięcie filmów trwających ponad 5 godzin
df.drop(df[df['runtime'] >= 300].index, inplace = True)
#usunięcie filmów trwających mniej niż 1 minutę
df.drop(df[df['runtime'] <= 1].index, inplace = True)
#usunięcie filmów bez oceny
df.drop(df[df['vote_count'] == 0].index, inplace = True)


#usunięcie filmów które wyszły przed 1960 rokiem
dates=[]
for element in df['release_date']:
    dates.append(datetime.strptime(element, '%Y-%m-%d').date().year)


df['release_date']=dates

df = df.dropna()
print(type(df['release_date'][0]))

df.drop(df[df['release_date'] <1960].index, inplace = True)

print("liczba obiektów: ", df.count(0)[0])
```

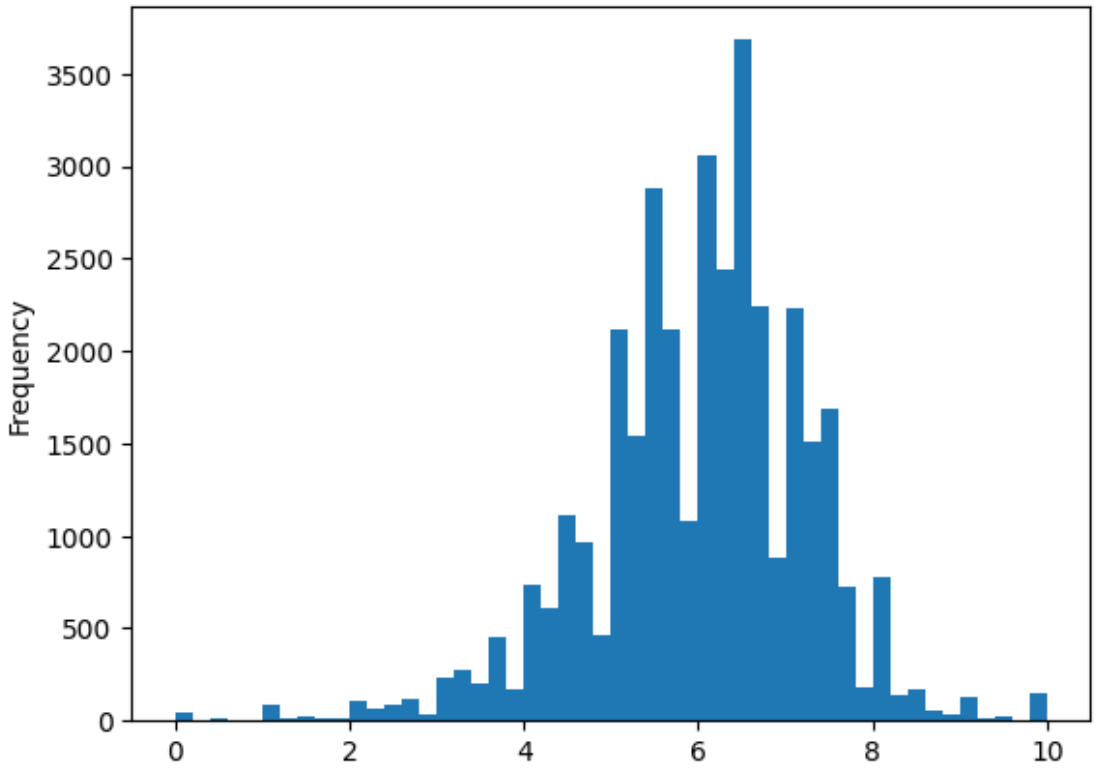
```
liczba obiektów:  44430
<class 'numpy.int64'>
liczba obiektów:  35712
```

Klasami w zestawie danych są gatunki, a cechy to: opis, data wydania, średnia ocen, ilość ocen. W danych źródłowych mamy 45466 obiektów, lecz po wyrzuceniu brakujących danych, skrajnych danych (np. filmy które trwają ponad 5 godzin) i ustawieniu daty najstarszych filmów na 1960 rok zostało nam ich jedynie 35712.

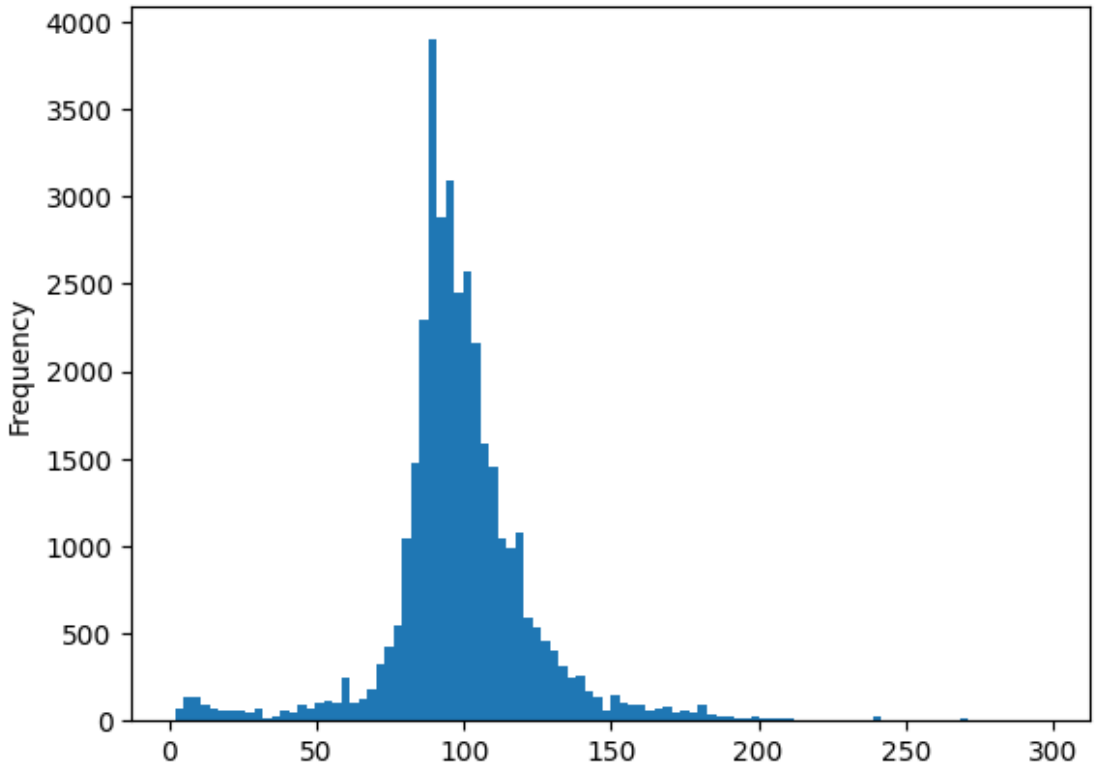
```
#Wykres, średnia i odchylenie standardowe oceny filmów
plt.show()
df['vote_average'].plot.hist(bins=50)
print("Average rating mean:"+str(df['vote_average'].mean())+' standard
deviation:'+str(df['vote_average'].std()))
print("Average rating varies from "+str(df['vote_average'].min())+" to
"+str(df['vote_average'].max()))
plt.show()
```

```
#Wykres, średnia i odchylenie standardowe czasu trwania filmów
df['runtime'].plot.hist(bins=100)
print("Runtime mean:"+str(df['runtime'].mean())+' standard
deviation:'+str(df['runtime'].std()))
print("Runtime varies from "+str(df['runtime'].min())+" to
"+str(df['runtime'].max()))
plt.show()

#Wykres, średnia i odchylenie standardowe ilości recenzji
df['vote_count'].plot.hist(bins=100,log=True)
print("Vote count mean:"+str(df['vote_count'].mean())+' standard
deviation:'+str(df['vote_count'].std()))
print("Vote count varies from "+str(df['vote_count'].min())+" to
"+str(df['vote_count'].max()))
plt.show()
print()
```

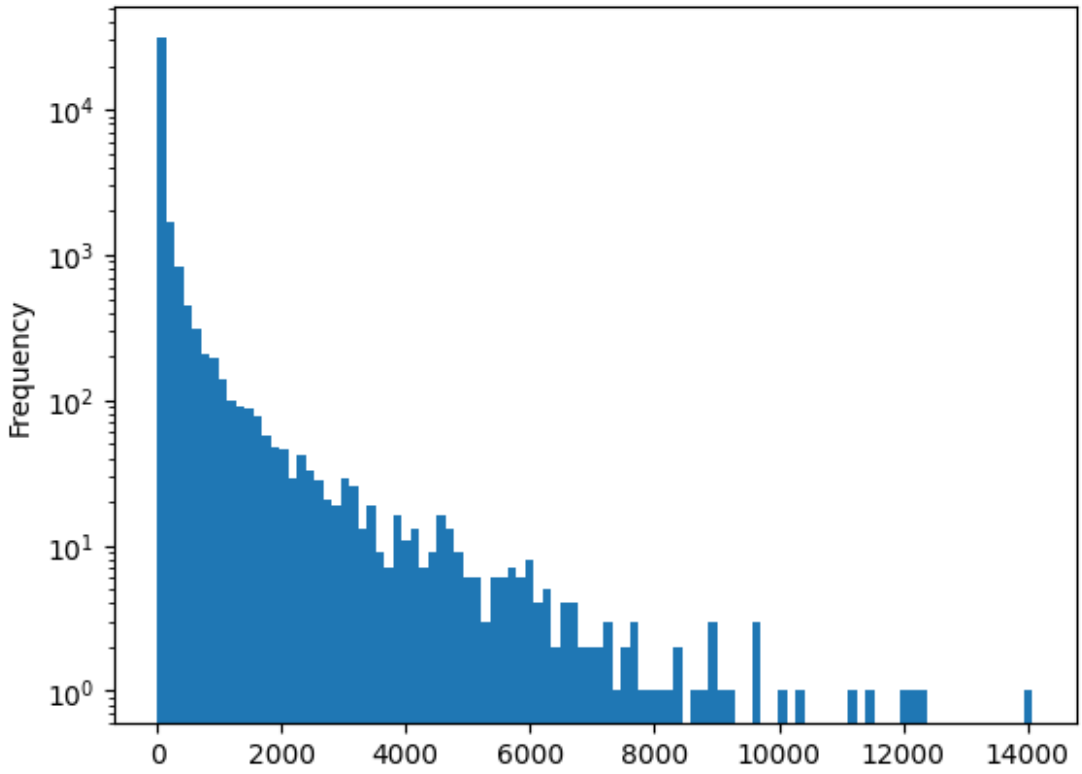
```
Average rating mean:5.99996639784925 standard
deviation:1.2554535618358154
Average rating varies from 0.0 to 10.0
```




```
Runtime mean:98.84828629032258 standard deviation:25.73440794579435
Runtime varies from 2.0 to 298.0
```




```
Vote count mean:136.07890905017922 standard
deviation:550.0630522213452
Vote count varies from 1.0 to 14075.0
```



Jak widać, po usunięciu niepotrzebnych danych wykres czasu trwania filmów znacząco zawężył się, po usunięciu filmów bez ocen zniknęły również filmy, które miały średnią ocen 0.

Kolejnym krokiem na drodze przefiltrowania i analizy zbioru danych wejściowych było zajęcie się kolumną "genres". Krok pierwszy - zamiana formatu a'la JSON, w którym były zapisane dane o gatunkach filmów na zwykłe listy.

```
import json
unique_genres = set()
genres_count = dict()
#obrabienie jsonów
def process_genres(genres):
    genres = genres[1:len(genres)-1]
    genres = genres.split("}, ")
    row_genres = []
    for genre in genres:
        if(len(genre)>1):
            if(genre[len(genre)-1] != "}"):
                genre += "}"
            genre = genre.replace("'", '"')
            genres_json = json.loads(genre)
            genre_name = genres_json["name"]
            row_genres.append(genre_name)
```

```
            if(genre_name in genres_count):
                genres_count[genre_name] +=1
            else:
                genres_count[genre_name] = 1
                unique_genres.add(genre_name)
    return row_genres
df2 = df.copy()
#zamiana na tablice gatunków
df2['genres'] = [process_genres(x) for x in df['genres']]
```

Następnie - przeanalizowanie jakie zestawy gatunków pojawiają się najczęściej i w jakiej ilości.

```
#liczenie filmów dla każdego gatunku
df2['genres_str'] = [''.join(x) for x in df['genres']]
df2.head(10)
unique_genres = dict()
def count_unique_genres(genre_name):
    if(genre_name in unique_genres):
        unique_genres[genre_name] +=1
    else:
        unique_genres[genre_name] = 1
[count_unique_genres(x) for x in df2['genres_str']]
unique_genres_keys_sorted = list(dict(sorted(unique_genres.items(),
key=lambda x:x[1], reverse=True)).keys())
for i in range(0, 20):
    print(str(unique_genres_keys_sorted[i]) + ' : ' + str(+
unique_genres[unique_genres_keys_sorted[i]]) + '\n')
```

```
[{'id': 18, 'name': 'Drama'}] : 3873

[{'id': 35, 'name': 'Comedy'}] : 2654

[{'id': 99, 'name': 'Documentary'}] : 2122

[] : 1145

[{'id': 18, 'name': 'Drama'}, {'id': 10749, 'name': 'Romance'}] : 1015

[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'name': 'Drama'}] : 965

[{'id': 27, 'name': 'Horror'}] : 891

[{'id': 35, 'name': 'Comedy'}, {'id': 10749, 'name': 'Romance'}] : 705

[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'name': 'Drama'}, {'id':
10749, 'name': 'Romance'}] : 517

[{'id': 27, 'name': 'Horror'}, {'id': 53, 'name': 'Thriller'}] : 506
```

```
[{'id': 18, 'name': 'Drama'}, {'id': 35, 'name': 'Comedy'}] : 439

[{'id': 18, 'name': 'Drama'}, {'id': 53, 'name': 'Thriller'}] : 425

[{'id': 53, 'name': 'Thriller'}] : 381

[{'id': 28, 'name': 'Action'}, {'id': 53, 'name': 'Thriller'}] : 284

[{'id': 18, 'name': 'Drama'}, {'id': 10769, 'name': 'Foreign'}] : 261

[{'id': 80, 'name': 'Crime'}, {'id': 18, 'name': 'Drama'}] : 260

[{'id': 28, 'name': 'Action'}] : 250

[{'id': 10749, 'name': 'Romance'}, {'id': 18, 'name': 'Drama'}] : 239

[{'id': 28, 'name': 'Action'}, {'id': 35, 'name': 'Comedy'}] : 221

[{'id': 18, 'name': 'Drama'}, {'id': 36, 'name': 'History'}] : 211
```

Ważnym elementem analizy było usunięcie filmów, które nie miały do siebie przypisanego żadnego gatunku, początkowo gatunek był w formacie json, przez co filmy bez gatunku miały puste stringi jako swoją wartość, a nie brak wartości, co uniemożliwiło przefiltrowanie takich filmów na początku. W celu bardziej skutecznej klasyfikacji konieczne było dokonanie pewnych manipulacji i uproszczeń na atrybucie decyzyjnym, czyli kolumnie "genres", w której wiele obiektów miało przypisaną więcej niż jedną wartość. Musieliśmy odrzucić gatunki mało popularne. Trzeba było też scalić wartości, które logicznie były blisko siebie i rozróżnianie ich w klasyfikacji miało małą szansę powodzenia - na przykład para gatunków "thriller" i "kryminał". Aby ocenić, które gatunki można ze sobą scalić przeprowadziliśmy analizę opisów filmów, tak aby móc scalić gatunki, które mają najpodobniejsze do siebie opisy.

```
#usuwanie filmów bez gatunku
df2.drop(df2[df2['genres'].map(lambda x: len(x) == 0)].index,
inplace=True)

print("liczba obiektów: ", df2.count(0)[0])

#zostawianie jednego gatunku dla filmu
main_genres=[]

for element in df2['genres']:
    main_genres.append(element[0])


df2['genres']=main_genres
```

```
print("liczba obiektów: ", df2.count(0)[0])
unique_genres = dict()
[count_unique_genres(x) for x in df2['genres']]
unique_genres_keys_sorted = list(dict(sorted(unique_genres.items(),
key=lambda x:x[1], reverse=True)).keys())
wanted_genres = unique_genres_keys_sorted[:10]
df2.drop(df2[df2['genres'].map(lambda x: x not in
wanted_genres)].index, inplace=True)

print("liczba obiektów: ", df2.count(0)[0])
print(wanted_genres)
```

```
liczba obiektów:  34567
liczba obiektów:  34567
liczba obiektów:  31113
['Drama', 'Comedy', 'Action', 'Documentary', 'Horror', 'Thriller',
'Adventure', 'Crime', 'Animation', 'Romance']
```

W celu analizy tekstu musieliśmy dokonać tokenizacji opisów, czyli rodzielić je na pojedyncze słowa, zamienić wszystkie litery na małe oraz usunąć niepotrzebne znaki przylegające do słów (takie jak nawiasy, czy cudzysłowia).

```
#tokenizacja
def tokenize(words):
    words = words.translate(str.maketrans('', '', string.punctuation))

    words = words.lower()
    words = words.split()
    return words

df2['overview'] = [tokenize(x) for x in df2['overview']]
print(df2['overview'][2])
```

```
['a', 'family', 'wedding', 'reignites', 'the', 'ancient', 'feud',
'between', 'nextdoor', 'neighbors', 'and', 'fishing', 'buddies',
'john', 'and', 'max', 'meanwhile', 'a', 'sultry', 'italian',
'divorcée', 'opens', 'a', 'restaurant', 'at', 'the', 'local', 'bait',
'shop', 'alarming', 'the', 'locals', 'who', 'worry', 'shell', 'scare',
'the', 'fish', 'away', 'but', 'shes', 'less', 'interested', 'in',
'seafood', 'than', 'she', 'is', 'in', 'cooking', 'up', 'a', 'hot',
'time', 'with', 'max']
```

Utworzyliśmy słownik, w którym przechowujemy listy słów dla każdego rozpatrywanego gatunku. Będzie on użyteczny w analizie danych tekstowych.

```
#tworzenie słowników, które zawierają słowa występujące dla
określonych gatunków
wordlists = dict()
```

```
for genre in wanted_genres:
    series = df2[df2['genres'] == genre]
    wordlists[genre] = dict()
    for words in series['overview']:
        for word in words:
            if(word in wordlists[genre]):
                wordlists[genre][word] +=1
            else:
                wordlists[genre][word] = 1
    wordlists[genre] = dict(sorted(wordlists[genre].items(),
key=lambda x:x[1], reverse=True))
for genre in wordlists.keys():
    for i, key in zip(range(0, 20), wordlists[genre].keys()):
        print(str(key) + " : " + str(wordlists[genre][key]) + ", ")
    print("\n")
```

```
the : 28001,
a : 22370,
and : 16748,
to : 15893,
of : 15009,
in : 11554,
his : 9023,
is : 8231,
her : 6593,
with : 5729,
he : 4810,
for : 4041,
an : 3711,
on : 3651,
who : 3405,
by : 3248,
as : 3177,
that : 3146,
their : 3021,
she : 2912,


the : 17972,
a : 15017,
to : 11776,
and : 10954,
of : 9094,
in : 7008,
his : 6293,
is : 5578,
with : 3919,
he : 3539,
her : 3268,
for : 2984,
```

on : 2583,
an : 2467,
their : 2401,
that : 2264,
who : 2210,
as : 2206,
by : 1962,
when : 1878,

the : 13749,
a : 9266,
to : 7328,
and : 6499,
of : 6355,
in : 4019,
his : 3681,
is : 3612,
he : 2067,
with : 2066,
an : 1739,
for : 1711,
by : 1646,
on : 1591,
who : 1423,
that : 1348,
as : 1299,
from : 1163,
when : 1152,
their : 1151,

the : 11601,
of : 6600,
and : 6351,
a : 4480,
to : 3402,
in : 3293,
is : 1529,
with : 1305,
on : 1298,
his : 1250,
as : 1187,
that : 1060,
for : 1049,
from : 1048,
film : 1047,
documentary : 929,
an : 924,
by : 886,

their : 885,
this : 877,

the : 7189,
a : 5828,
to : 3741,
of : 3521,
and : 3350,
in : 2185,
is : 1755,
his : 1431,
her : 1312,
by : 1017,
that : 999,
with : 998,
an : 988,
on : 903,
for : 878,
their : 798,
they : 787,
he : 734,
as : 706,
who : 686,

the : 4115,
a : 3593,
to : 2433,
and : 2052,
of : 1979,
in : 1444,
is : 1317,
his : 1076,
her : 857,
with : 715,
he : 679,
on : 600,
for : 585,
that : 579,
an : 568,
by : 483,
who : 436,
as : 418,
when : 403,
she : 392,

the : 4423,
a : 2704,

to : 2307,
and : 2164,
of : 1936,
in : 1220,
is : 956,
his : 955,
with : 612,
on : 564,
he : 536,
by : 494,
an : 490,
for : 476,
that : 445,
who : 394,
their : 387,
her : 387,
from : 379,
as : 378,

the : 3164,
a : 2805,
to : 1744,
and : 1728,
of : 1716,
in : 1165,
his : 957,
is : 950,
he : 516,
with : 513,
for : 446,
her : 445,
an : 388,
on : 388,
who : 379,
by : 373,
that : 344,
as : 333,
from : 290,
their : 277,

the : 3561,
a : 2111,
and : 1834,
to : 1684,
of : 1575,
in : 954,
is : 727,
his : 719,

```
with : 484,
he : 410,
on : 392,
by : 388,
that : 373,
for : 362,
an : 349,
as : 348,
her : 331,
from : 304,
their : 293,
who : 266,


the : 2322,
a : 2204,
and : 1695,
to : 1620,
of : 1161,
in : 1051,
is : 874,
her : 801,
his : 752,
with : 626,
he : 498,
for : 444,
she : 411,
who : 385,
an : 382,
on : 342,
love : 339,
that : 328,
as : 304,
their : 291,
```

Zdecydowaliśmy się spróbować uczenia maszynowego dla atrybutów ilościowych oddzielnie - tutaj uczenie naiwnym klasyfikatorem Bayesa.

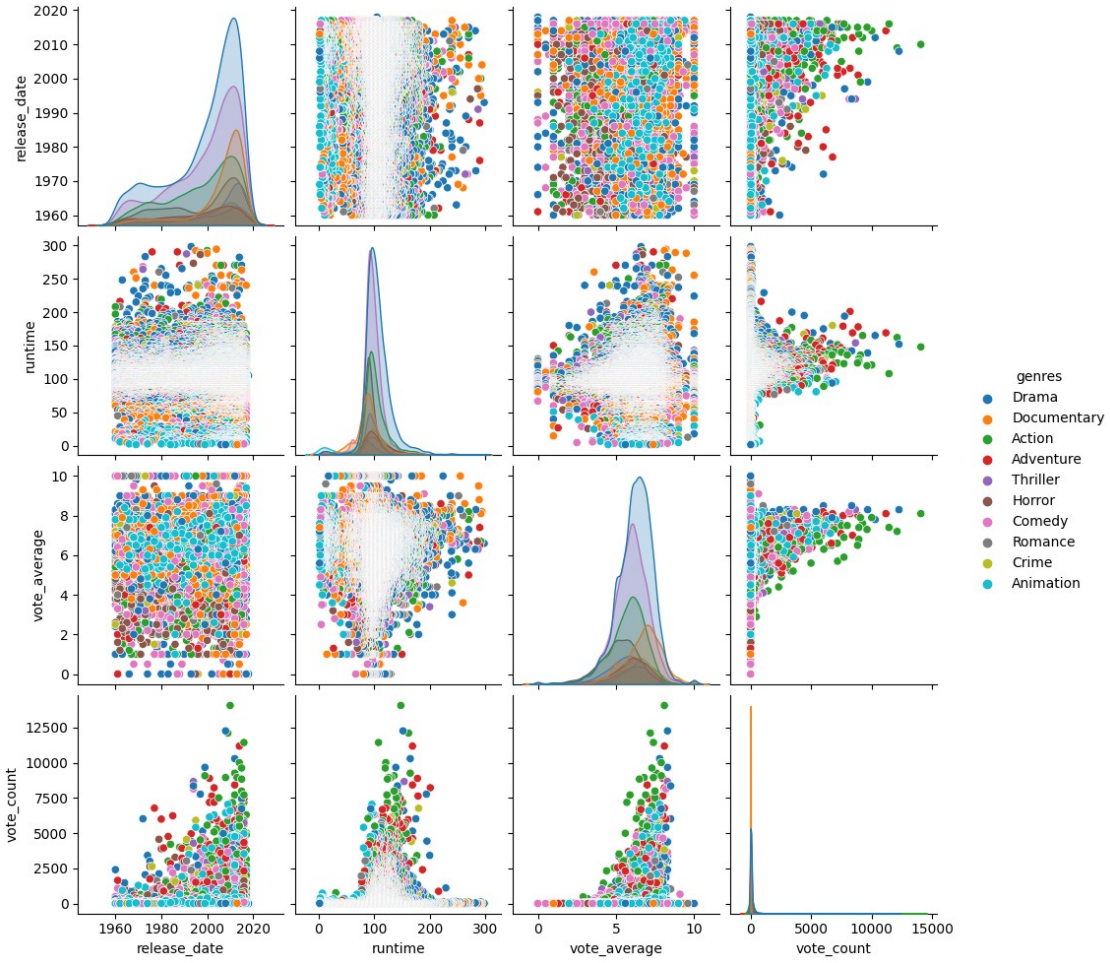
```
#macierze pomyłek
from sklearn.naive_bayes import GaussianNB
df2 = df2.dropna()
sns.pairplot(df2, kind="scatter", hue = "genres")
def split(df,proporcja):
    opis_ucz, opis_test, dec_ucz, dec_test =
train_test_split(df.iloc[:,1:],
df.iloc[:,0].astype('category').cat.codes, test_size=proporcja)#,
random_state=0)
    return {"opis_ucz":opis_ucz, "opis_test":opis_test,
```

```
"dec_ucz":dec_ucz, "dec_test":dec_test}
def verify(model,dane,atryb):
    model.fit(dane["opis_ucz"].iloc[:,atryb], dane["dec_ucz"])
    wynik_ucz = model.predict(dane["opis_ucz"].iloc[:,atryb])
    wynik_test = model.predict(dane["opis_test"].iloc[:,atryb])
    mp = confusion_matrix(dane["dec_ucz"],wynik_ucz)
    print("macierz pomyłek - zbiór uczący,
dokładność:",np.sum(np.diag(mp))/np.sum(mp))
    learn_set_score =
model.score(dane['opis_ucz'].iloc[:,atryb] ,dane['dec_ucz'])

print(model.score(dane['opis_ucz'].iloc[:,atryb] ,dane['dec_ucz']))
    print(mp)
    mp = confusion_matrix(dane["dec_test"],wynik_test)
    print("macierz pomyłek - zbiór testowy,
dokładność:",np.sum(np.diag(mp))/np.sum(mp))
    test_set_score =
model.score(dane['opis_test'].iloc[:,atryb] ,dane['dec_test'])

print(model.score(dane['opis_test'].iloc[:,atryb] ,dane['dec_test']))
    print(mp)
    return (learn_set_score, test_set_score)

model = GaussianNB()
try_count = 5
learn_set_score = 0
test_set_score = 0
for k in range(0, try_count):
    data = split(df2, 0.3)
    atryb = [3, 5, 6]
    learn_score, test_score = verify(model, data, atryb)
    learn_set_score += learn_score
    test_set_score += test_score
print('Średnia dokładność z ' + str(try_count) + ' prób wynosi dla
zbioru uczącego ' + str(learn_set_score / try_count) + ' , a dla
zbioru testowego ' + str(test_set_score / try_count))

macierz pomyłek - zbiór uczący, dokładność: 0.17039349832407366
0.17039349832407366
[[  89   93    6  383    0 1682  372  129    0    0]
 [  45   70    7  133    0  443  142   29    0    0]
 [  23   18   27   94    0  470   39    4    0    0]
 [  92   25    3  740    0 3417  416  136    0    0]
 [  20    8    1   93    0  515  129    9    0    0]
 [   0    0    2   39    0 1814   40   31    0    0]
 [  68   67    6  427    0 5164  829  120    0    0]
 [  33   10    3  318    0 1066   90  142    0    0]
 [   6    7    1   51    0  454   58   16    0    0]
 [  18   19    1  152    0  678   91   56    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.17580887079494323
```

```
0.17580887079494323
[[  36   40    2  182    0  733  151   62    0    0]
 [  21   25    3   56    0  181   60   11    0    0]
 [  10    9   18   39    0  194   20    1    0    0]
 [  23   11    1  347    0 1481  160   60    0    0]
 [  10   10    0   32    0  255   66    9    0    0]
 [   0    1    2   15    0  753   18    7    0    0]
 [  24   16    3  190    0 2158  386   38    0    0]
 [  15    3    1  134    0  456   38   76    0    0]
 [   5    3    0   22    0  207   24    6    0    0]
 [  13    6    1   58    0  262   49   25    0    0]]
macierz pomyłek - zbiór uczący, dokładność: 0.17245970889388862
0.17245970889388862
[[  94   83    3  517    0 1629  309  152    0    0]
 [  45   59    4  157    0  433  127   33    0    0]
 [  26   14   28  112    0  483   37    3    0    0]
 [  87   15    1  890    0 3343  367  145    0    0]
 [  25   13    1  102    0  528  128   15    0    0]
 [   0    0    0   61    0 1794   24   30    0    0]
 [  70   54    0  570    0 5101  726  124    0    0]
 [  32    7    0  372    0  985   84  165    0    0]
 [   8    5    1   63    0  432   48   14    0    0]
 [  28   13    0  192    0  619   89   60    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.17859438611527748
0.17859438611527748
[[  54   28    2  234    0  673  128   54    0    0]
 [  33   24    3   57    0  188   51   12    0    0]
 [  12    8   14   34    0  175   17    3    0    0]
 [  41    8    2  393    0 1393  164   63    0    0]
 [   8    2    0   44    0  233   53    5    0    0]
 [   0    0    0   22    0  769   13    9    0    0]
 [  31   24    1  241    0 2171  328   55    0    0]
 [  20    2    0  184    0  411   38   85    0    0]
 [   6    2    0   25    0  222   22   12    0    0]
 [   7    7    0   79    0  275   33   27    0    0]]
macierz pomyłek - zbiór uczący, dokładność: 0.1851783828458607
0.1851783828458607
[[  96   84    6  455    0 1509  427  161    0    0]
 [  47   62    6  151    0  397  162   27    0    0]
 [  32   17   24   96    0  474   44    6    0    0]
 [  96   17    2  866    0 3256  465  156    0    0]
 [  22   10    0   99    0  513  146   14    0    0]
 [   0    1    1   53    0 1752   63   29    0    0]
 [  78   59    6  544    0 4783 1048  144    0    0]
 [  50    5    3  350    0  997   87  185    0    0]
 [  10    6    0   53    0  432   68   19    0    0]
 [  28   16    1  180    0  611  110   62    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.1917720162845511
0.1917720162845511
[[  54   35    2  225    0  656  172   78    0    0]
```

```
 [  30   31    4   59    0  170   62   18    0    0]
 [  11    6   20   41    0  181   14    0    0    0]
 [  47   11    1  360    0 1396  163   76    0    0]
 [  12    7    1   40    0  214   72    7    0    0]
 [   0    1    5   22    0  763   22   10    0    0]
 [  37   27    0  233    0 1999  477   61    0    0]
 [   8    5    1  151    0  417   41   85    0    0]
 [   5    3    1   30    0  185   42    6    0    0]
 [   9    4    1   64    0  259   51   33    0    0]]
macierz pomyłek - zbiór uczący, dokładność: 0.17507690894898756
0.17507690894898756
[[ 114   81    5  469    0 1616  339  167    0    0]
 [  61   60    7  147    0  435  125   36    0    0]
 [  35   17   31   83    0  475   45    2    0    0]
 [ 115   13    2  812    0 3199  455  155    0    0]
 [  23   12    1   87    0  514  152   18    0    0]
 [   0    0    0   46    0 1790   40   28    0    0]
 [  84   51    3  468    0 5086  820  145    0    0]
 [  49    7    0  357    0 1002   97  186    0    0]
 [  13    6    1   59    0  461   55   18    0    0]
 [  32   13    0  171    0  618   92   73    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.17934433254767518
0.17934433254767518
[[  55   23    1  203    0  670  152   65    0    0]
 [  29   19    0   54    0  178   64   11    0    0]
 [  11    5    9   45    0  187   18    3    0    0]
 [  38    6    1  357    0 1516  172   71    0    0]
 [  12    3    0   31    0  237   63    4    0    0]
 [   0    0    0   15    0  771   20   12    0    0]
 [  38   25    1  203    0 2139  379   54    0    0]
 [  12    2    0  169    0  390   30   84    0    0]
 [   5    0    0   18    0  195   23    6    0    0]
 [   7    4    0   72    0  272   52   23    0    0]]
macierz pomyłek - zbiór uczący, dokładność: 0.17448000367326325
0.17448000367326325
[[ 107   77    6  372    0 1643  347  260    0    0]
 [  50   52    7  128    0  425  134   55    0    0]
 [  29   12   33   99    0  448   40   11    0    0]
 [  95   18    1  737    0 3280  415  266    0    0]
 [  27   11    1   89    0  532  148   26    0    0]
 [   0    0    2   43    0 1789   37   42    0    0]
 [  68   54    2  413    0 5030  792  219    0    0]
 [  32    3    2  302    0  960   88  290    0    0]
 [  11    4    1   53    0  484   58   31    0    0]
 [  28   14    1  147    0  604   98   96    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.1742018427255196
0.1742018427255196
[[  40   32    2  164    0  656  145  109    0    0]
 [  30   28    3   53    0  181   62   18    0    0]
 [   9   10   12   31    0  214   17    1    0    0]
```

```
 [  31    4    2  305    0 1485  170  103    0    0]
 [   6    4    0   30    0  223   47   13    0    0]
 [   0    0    2   19    0  760    9   19    0    0]
 [  30   20    5  188    0 2212  361  102    0    0]
 [  20    6    2  125    0  397   38  120    0    0]
 [   2    3    0   19    0  168   18    8    0    0]
 [   7    5    0   59    0  288   30   52    0    0]]
Średnia dokładność z 5 prób wynosi dla zbioru uczącego
0.17551770053721477 , a dla zbioru testowego 0.1799442896935933
```



Podobne macierze, tym razem porównane z klasyfikatorem k-nn, dla różnych ilości sąsiadów.

```python
for k in range(5, 80, 5):
    print('kneighbours : ' + str(k))
    model = KNeighborsClassifier(k)
    data = split(df2, 0.3)
    atryb = [3, 5, 6]
    verify(model, data, atryb)
```

```
kneighbours : 5
macierz pomyłek - zbiór uczący, dokładność: 0.4833555259653795
```

```
0.4833555259653795
[[1404   24   32  449   22   66  635   83    9   17]
 [ 191  208   26  149    4   30  214   31    1    8]
 [  93   37  333   94    1   56   75   15    1    5]
 [ 566   73  103 2725   20  177  968  124    6   29]
 [ 153   29   12  179  104   39  264   15    3    2]
 [ 151   24   64  348    8  854  427   22    3    2]
 [ 602   99   87 1103   60  309 4340   81   15   30]
 [ 326   37   42  421    6   48  326  432    3   11]
 [ 115   23   14  140    2   29  237   20   32    4]
 [ 196   31   20  254   14   31  294   41    2   95]]
macierz pomyłek - zbiór testowy, dokładność: 0.27951574887508035
0.27951574887508035
[[ 285   36   23  318    6   47  426   59    6   13]
 [  67   16   17   85    4   16  132   23    1    3]
 [  40    8   76   52    2   27   44    6    0    1]
 [ 365   53   43  687   23  151  681   94   10   14]
 [  65    5    6   97   10   17  144    8    2    3]
 [  68   26   34  175    2  233  270    7    2    2]
 [ 426   62   41  712   45  186 1200   71    9   18]
 [ 163   10   23  253    5   23  149   97    3    7]
 [  38    5    3   77    0   23   89    6    0    3]
 [  99    9    8  119    6   16  163   25    1    5]]
kneighbours : 10
macierz pomyłek - zbiór uczący, dokładność: 0.4321135038339685
0.4321135038339685
[[ 885   27   38  633    9   73  965  103    1   11]
 [ 153   85   29  208    6   33  306   33    0    4]
 [  57   17  273  142    4   75   95   21    0    2]
 [ 417   35   72 2492   11  225 1429  126    2   17]
 [ 112   12    6  231   45   31  352   23    0    4]
 [  94    8   45  359    9  761  615   19    0    1]
 [ 478   36   55 1193   29  287 4490   75    3    9]
 [ 229   21   27  591    8   52  376  347    2    6]
 [  77    7   13  161    6   32  307   17    1    1]
 [ 147   10   14  314    7   29  389   59    1   32]]
macierz pomyłek - zbiór testowy, dokładność: 0.3034068995071781
0.3034068995071781
[[ 179   24   24  416    4   39  473   49    0    7]
 [  72   12   15  104    3   10  141   10    0    2]
 [  27    6   87   74    0   32   46    8    0    0]
 [ 249   11   45  737   12  118  818   83    1   12]
 [  34    6    5   98    3   12  174    8    0    1]
 [  47    2   27  181    1  215  330    6    0    2]
 [ 267   26   44  746   19  166 1523   37    1   12]
 [ 123   12   19  301    3   19  170   73    0    6]
 [  31    3    2   65    1   16  116    3    0    1]
 [  58    6    3  148    1   12  171   25    0    3]]
kneighbours : 15
macierz pomyłek - zbiór uczący, dokładność: 0.4118187244593416
```

```
0.4118187244593416
[[ 664   31   43  761    6   75 1120  109    0   10]
 [ 141   64   29  254    0   30  325   25    0    4]
 [  50   16  249  157    0   77  122   10    0    1]
 [ 359   18   68 2370    2  243 1644  131    0   10]
 [  96    8    8  230   11   31  423   23    0    1]
 [  55    2   33  350    1  693  732   15    0    2]
 [ 345   20   67 1232   13  252 4614   66    1    4]
 [ 192    5   25  654    3   46  452  286    0    4]
 [  56    4    9  151    0   26  325   13    1    0]
 [ 102   10   13  325    5   23  441   45    0   17]]
macierz pomyłek - zbiór testowy, dokładność: 0.3229055067495179
0.3229055067495179
[[ 164   13   20  352    4   35  498   52    0    3]
 [  47   10    6  108    2   23  145   13    0    0]
 [  19    5   90   66    0   37   54   13    0    0]
 [ 189    8   43  731    3  120  894   75    0    4]
 [  38    3    3   95    1   11  166    8    0    1]
 [  28    0   27  170    0  234  372    7    0    1]
 [ 228   16   35  687   10  154 1701   47    0    4]
 [  83    4   19  318    0   27  182   82    0    3]
 [  26    3    3   72    1   22  145    3    0    0]
 [  57    3    3  154    1   16  194   19    0    1]]
kneighbours : 20
macierz pomyłek - zbiór uczący, dokładność: 0.39776849258459984
0.39776849258459984
[[ 514   18   39  763    0   71 1263   97    1    2]
 [ 119   39   26  253    1   39  357   31    1    0]
 [  49    6  243  148    0   87  136   15    1    2]
 [ 270   11   74 2153    1  274 1862  159    0    0]
 [  62    8    6  212    1   34  469   20    0    0]
 [  36    1   41  351    1  695  789   11    1    0]
 [ 282   15   69 1220    4  288 4722   65    1    1]
 [ 146    4   31  696    1   56  458  291    0    1]
 [  35    2   10  132    0   34  351   15    1    0]
 [  86    4    7  332    0   31  465   55    1    4]]
macierz pomyłek - zbiór testowy, dokładność: 0.33769016498821514
0.33769016498821514
[[ 147   15   23  360    0   31  557   58    1    0]
 [  57    7   15  107    0    8  157    7    0    2]
 [  21    6   93   64    0   42   48    5    0    0]
 [ 149    5   39  779    4  132  914   84    0    2]
 [  40    3    5   89    1    8  191    8    0    0]
 [  14    0   19  169    0  245  346    2    1    0]
 [ 137   11   27  669    4  138 1802   41    0    0]
 [  77    1   13  293    0   23  215   78    0    1]
 [  16    3    2   71    0   16  169    3    0    0]
 [  36    3    5  144    0   17  219   19    1    0]]
kneighbours : 25
macierz pomyłek - zbiór uczący, dokładność: 0.3924422608935213
```

```
0.3924422608935213
[[ 413   22   41  835    0   66 1256  121    0    2]
 [ 101   33   22  265    0   33  351   39    0    1]
 [  37    7  242  176    0   70  116   18    0    1]
 [ 225    8   75 2174    0  249 1993  142    0    3]
 [  51    5    8  209    0   22  506   23    0    1]
 [  21    0   39  358    1  642  829   13    0    0]
 [ 210   17   70 1252    3  277 4768   71    0    3]
 [ 123    7   32  718    0   47  468  273    0    2]
 [  34    2   11  149    0   27  353   14    0    0]
 [  72    3    8  332    0   21  494   51    0    2]]
macierz pomyłek - zbiór testowy, dokładność: 0.3375830297835869
0.3375830297835869
[[ 112   12   18  394    1   33  586   47    0    1]
 [  47    7   13  116    0   14  170   14    0    0]
 [  18    6   88   80    0   52   51    3    0    1]
 [ 127    5   40  780    1  107  900   82    0    1]
 [  17    2    3   94    0   13  197    5    0    1]
 [  11    0   27  151    0  241  381    8    0    0]
 [ 125    6   38  663    1  121 1829   40    0    2]
 [  64    3   11  308    0   20  214   94    0    1]
 [  13    2    2   65    0   10  174    4    0    0]
 [  33    4    4  154    0   11  215   25    0    0]]
kneighbours : 30
macierz pomyłek - zbiór uczący, dokładność: 0.38661095550759905
0.38661095550759905
[[ 405   20   46  811    0   68 1310  115    0    2]
 [ 107   28   25  257    0   34  404   28    0    0]
 [  30   11  219  182    0   84  133   21    0    1]
 [ 205   10   74 2089    1  262 2028  123    0    1]
 [  44    7    4  228    0   20  480   15    0    1]
 [  23    0   44  332    0  646  867   14    0    0]
 [ 191   15   67 1232    0  289 4779   77    0    2]
 [ 105    5   29  717    0   66  485  254    1    2]
 [  27    0   10  137    0   36  378   11    0    0]
 [  81    8    3  344    0   33  488   48    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.33597600171416325
0.33597600171416325
[[  82   17   17  404    0   33  590   40    0    0]
 [  39    7    7  103    1   12  160   14    0    0]
 [  12    6   82   80    0   41   62    2    0    0]
 [  88    6   25  760    0  127 1037   76    0    0]
 [  17    1    2  103    0   12  214    9    0    0]
 [   9    0   19  145    0  254  364    5    0    0]
 [ 114   14   25  642    0  137 1876   35    0    1]
 [  59    1    9  332    0   23  223   74    0    0]
 [  10    5    3   71    0   12  157    3    0    0]
 [  23    0    4  146    0    8  216   26    0    1]]
kneighbours : 35
macierz pomyłek - zbiór uczący, dokładność: 0.3862436291840764
```

```
0.3862436291840764
[[ 344    5   41  856    0   59 1366  100    0    0]
 [ 105    4   23  268    0   30  370   32    0    0]
 [  39    4  212  173    0   92  131   20    0    0]
 [ 206    1   69 2130    0  246 2073  128    0    0]
 [  47    1    5  208    0   28  479   13    0    0]
 [  21    0   45  332    0  648  853   15    0    0]
 [ 182    7   57 1241    1  275 4838   75    0    0]
 [ 109    0   24  766    0   57  478  234    0    0]
 [  25    0    8  159    0   38  375    5    0    0]
 [  72    2    5  326    1   30  517   48    0    2]]
macierz pomyłek - zbiór testowy, dokładność: 0.3365116777373045
0.3365116777373045
[[ 111    0   15  377    0   39  592   55    0    0]
 [  52    6   10  120    1   16  178   11    0    0]
 [  16    2   83   84    0   38   54   18    0    0]
 [  97    1   29  768    0  119  990   55    0    0]
 [  24    0    2  117    0    6  220    7    0    0]
 [   8    0   19  152    0  223  401    5    0    0]
 [ 103    1   34  607    1  147 1885   42    0    0]
 [  59    1   13  335    0   26  218   65    0    0]
 [  13    1    3   62    0    8  161    2    0    0]
 [  25    0    4  149    1    9  211   27    0    0]]
kneighbours : 40
macierz pomyłek - zbiór uczący, dokładność: 0.37981541852242984
0.37981541852242984
[[ 320    4   44  856    0   79 1353  111    0    0]
 [ 109    7   20  270    0   35  374   34    0    0]
 [  34    3  205  205    0   94  123   11    0    0]
 [ 151    2   69 2145    0  279 2103  117    0    0]
 [  49    2    9  232    0   25  509   12    0    0]
 [  13    0   44  382    0  633  804    9    0    0]
 [ 188    5   70 1284    0  299 4727   83    0    0]
 [ 101    1   27  785    0   50  469  235    0    0]
 [  29    1    4  162    0   28  376    4    0    0]
 [  54    2    6  342    0   31  479   57    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.3379044353974716
0.3379044353974716
[[  80    2   16  417    0   26  610   42    0    0]
 [  44    0    8  128    0   16  171   10    0    0]
 [  16    1   80   93    0   36   58    7    0    0]
 [  96    0   27  762    0  129  971   60    0    1]
 [  13    1    1   89    0   10  201    4    0    0]
 [   7    0   18  130    0  253  423    6    0    0]
 [  84    2   26  646    0  144 1912   26    0    0]
 [  43    0   11  338    0   33  225   67    0    0]
 [  11    0    6   63    0   12  158    6    0    0]
 [  27    0    3  169    0    8  230   21    0    0]]
kneighbours : 45
macierz pomyłek - zbiór uczący, dokładność: 0.3780247026952569
```

```
0.3780247026952569
[[ 295    9   47  832    0   64 1436   97    0    0]
 [ 106   12   22  280    0   40  376   27    0    0]
 [  42    5  225  186    0   75  127   20    0    0]
 [ 145    4   73 2027    0  238 2210  102    0    0]
 [  46    1    7  211    0   23  496   10    0    0]
 [   9    0   56  319    0  590  975    8    0    0]
 [ 162    8   84 1212    0  258 4867   58    0    0]
 [  72    3   28  788    0   62  502  217    0    0]
 [  18    0    7  153    0   33  385   10    0    0]
 [  53    3   10  319    0   26  525   43    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.3477608742232698
0.3477608742232698
[[  91    8   20  405    0   32  580   44    0    0]
 [  56    0    8  107    0   11  169   12    0    0]
 [  15    2   79   83    0   46   56    5    0    0]
 [  84    8   42  786    0  119 1003   71    0    0]
 [  13    2    3   95    0    9  236    5    0    0]
 [   2    0   24  129    0  223  385    2    0    0]
 [  76    3   30  585    0  128 1991   34    0    0]
 [  51    1   12  339    0   25  209   76    0    0]
 [   6    0    6   53    0   13  175    1    0    0]
 [  20    3    2  149    0   10  245   21    0    0]]
kneighbours : 50
macierz pomyłek - zbiór uczący, dokładność: 0.3744891868313513
0.3744891868313513
[[ 236   18   39  875    0   72 1443   79    0    0]
 [  81   20   12  310    0   30  374   22    0    0]
 [  30    7  230  213    0   77  126   19    0    0]
 [ 127    6   81 2089    0  228 2193  102    0    0]
 [  29    4    9  210    0   16  531   13    0    0]
 [   6    0   41  347    0  566  926   20    0    0]
 [ 136   15   79 1259    0  241 4815   72    0    0]
 [  73    2   25  789    0   49  537  200    0    0]
 [  15    2   11  158    0   33  403    9    0    0]
 [  49    6    5  354    0   26  516   43    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.34765373901864155
0.34765373901864155
[[  75   11   28  436    0   24  590   34    0    0]
 [  39    6   12  114    0   15  174   17    0    0]
 [  12    2   69   88    0   42   42    9    0    0]
 [  64    2   27  799    0   94 1043   57    0    0]
 [  13    2    2  102    0   16  206    4    0    0]
 [   3    0   19  169    0  213  401   11    0    0]
 [  63    5   28  599    0  116 2029   39    0    0]
 [  44    1   11  362    0   22  216   54    0    0]
 [   6    0    3   62    0    7  148    3    0    0]
 [  14    3    5  142    0   11  236   19    0    0]]
kneighbours : 55
macierz pomyłek - zbiór uczący, dokładność: 0.37178015519537166
```

```
0.37178015519537166
[[ 225    8   41  882    0   67 1511   87    0    0]
 [ 100   11   16  301    0   32  383   23    0    0]
 [  33    5  192  203    0   79  135   15    0    0]
 [ 117    1   61 1972    0  236 2298   91    0    0]
 [  33    0    7  213    0   21  512   12    0    0]
 [   5    0   38  351    0  569  922   17    0    0]
 [ 138    3   68 1189    0  242 4939   60    0    0]
 [  77    0   21  799    0   49  563  189    0    0]
 [  20    1   10  144    0   31  393   12    0    0]
 [  45    1    5  326    0   25  564   40    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.3488322262695522
0.3488322262695522
[[  89    7   13  377    0   27  584   42    0    0]
 [  35    3    3  106    0   16  185   12    0    0]
 [  13    0   81  104    0   46   52    8    0    0]
 [  83    1   29  770    0   95 1094   64    0    0]
 [  19    1    1   83    0   13  237    5    0    0]
 [   2    0   12  155    0  224  421    6    0    0]
 [  75    0   31  598    0  100 2028   25    0    0]
 [  28    0   12  334    0   32  220   61    0    0]
 [   5    0    3   61    0   12  165    3    0    0]
 [  19    1    2  166    0   11  208   16    0    0]]
kneighbours : 60
macierz pomyłek - zbiór uczący, dokładność: 0.3701271867395197
0.3701271867395197
[[ 221    9   50  899    0   58 1423   82    0    0]
 [ 100   10   13  308    0   29  375   24    0    0]
 [  31    5  202  227    0   88  126   10    0    0]
 [ 108    5   78 2061    0  242 2291   99    0    0]
 [  30    3    4  229    0   18  499   16    0    0]
 [   2    0   47  347    0  547  941    7    0    0]
 [ 129    5   81 1264    0  230 4869   55    0    0]
 [  74    1   27  828    0   43  555  151    0    0]
 [  18    1    9  148    0   21  390    9    0    0]
 [  40    3    5  339    0   27  561   32    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.3519391472037712
0.3519391472037712
[[  74   10   13  441    0   37  618   25    0    0]
 [  28    8    8  115    0   16  187    5    0    0]
 [  15    1   78  107    0   29   42    5    0    0]
 [  45    2   24  775    0   85 1056   41    0    0]
 [  12    1    3   94    0    8  239    1    0    0]
 [   3    0   19  170    0  225  413    1    0    0]
 [  61    2   24  573    0  108 2066   29    0    0]
 [  25    0   12  372    0   28  210   59    0    0]
 [   7    0    3   63    0   18  172    1    0    0]
 [  20    1    3  154    0    7  221   16    0    0]]
kneighbours : 65
macierz pomyłek - zbiór uczący, dokładność: 0.371137334129207
```

```
0.371137334129207
[[ 260    0   41  897    0   59 1443   78    0    1]
 [ 102    0   11  280    0   32  396   22    0    0]
 [  35    1  174  234    0   82  137   15    0    0]
 [ 121    0   55 1905    0  237 2394   83    0    0]
 [  30    0    2  200    0   25  510    9    0    1]
 [   5    0   35  340    0  579  958   13    0    0]
 [ 146    0   62 1241    0  240 5001   47    0    0]
 [  85    0   25  802    0   51  545  163    0    0]
 [  21    0    6  140    0   28  388    5    0    0]
 [  34    0    5  350    0   20  532   39    0    1]]
macierz pomyłek - zbiór testowy, dokładność: 0.34958217270194986
0.34958217270194986
[[  74    0   20  384    0   30  644   29    0    0]
 [  65    1    7  123    0   15  170    2    0    0]
 [  25    0   90   92    0   31   40   10    0    0]
 [  71    0   33  834    0  119 1020   40    0    0]
 [  24    0    2   91    0    8  249    6    0    0]
 [   3    0   18  128    0  215  427    1    0    0]
 [  57    0   24  565    0  107 1990   16    0    0]
 [  26    0   12  362    0   27  228   59    0    0]
 [   9    0    5   63    0   10  182    3    0    0]
 [  23    0    1  150    0   14  247   13    0    0]]
kneighbours : 70
macierz pomyłek - zbiór uczący, dokładność: 0.37031084990128105
0.37031084990128105
[[ 161    2   43  942    0   72 1438   72    0    0]
 [  89    3   16  275    0   36  400   16    0    0]
 [  36    2  196  211    0   93  123   10    0    0]
 [  77    0   61 1999    0  257 2377   85    0    0]
 [  22    0    2  214    0   25  514   14    0    0]
 [   2    0   40  325    0  625  916   11    0    0]
 [  99    0   78 1287    0  240 4917   48    0    0]
 [  49    0   21  837    0   66  565  164    0    0]
 [  10    0   11  144    0   29  395    5    0    0]
 [  32    0    6  346    0   29  564   35    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.3467966573816156
0.3467966573816156
[[  73    1   22  418    0   32  657   27    0    0]
 [  42    1    6  124    0   14  190   14    0    0]
 [  12    0   76  102    0   43   54    8    0    0]
 [  42    1   36  773    0  124 1039   41    0    0]
 [   9    0    3   99    0   11  241    3    0    0]
 [   1    0   15  144    0  224  414    5    0    0]
 [  45    0   30  570    0  131 2031   20    0    0]
 [  19    0   10  347    0   23  225   59    0    0]
 [   8    0    1   66    0   13  174    4    0    0]
 [  14    0    1  149    0    5  233   15    0    0]]
kneighbours : 75
macierz pomyłek - zbiór uczący, dokładność: 0.3665916708756141
```

```
0.3665916708756141
[[ 180    6   39  938    0   67 1501   58    0    0]
 [  90    5   19  284    0   26  414   17    0    0]
 [  31    1  177  241    0   78  122    8    0    0]
 [  94    3   66 1898    0  244 2452   83    0    0]
 [  32    1    5  189    0   27  556   11    0    0]
 [   1    0   42  334    0  556  935   10    0    0]
 [ 105    7   65 1173    0  262 5037   47    0    0]
 [  41    0   29  842    0   59  567  131    0    0]
 [  18    0    6  128    0   31  386    6    0    0]
 [  30    0    5  337    0   24  575   27    0    0]]
macierz pomyłek - zbiór testowy, dokładność: 0.3551532033426184
0.3551532033426184
[[  80    3   16  373    0   29  640   30    0    0]
 [  44    2    2  127    0   16  176    4    0    0]
 [  12    3   86  107    0   38   60    2    0    0]
 [  41    0   23  825    0   99 1063   21    0    0]
 [  11    1    2   94    0    7  218    3    0    0]
 [   0    0   14  148    0  257  422    3    0    0]
 [  60    2   28  551    0  118 2013   28    0    0]
 [  20    0    4  375    0   20  245   52    0    0]
 [   7    0    5   75    0   11  185    2    0    0]
 [  10    0    4  136    0    9  254   18    0    0]]
```

Jak widać, dla liczby sąsiadów k=40, klasyfikator działa w pewnym stopniu - ze skutecznością około 40%.

Kontynuując przygotowania do uczenia na danych tekstowych, zdecydowaliśmy się zlematyzować opisy filmów(Overview) w celu zmniejszenia worka słów oraz poprawienia wyników uczenia maszynowego.

```
#lematyzacja
from nltk.corpus import stopwords
from nltk.corpus import wordnet
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()

def checkWordType(word):    #Pomocnicza
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

def lematize(words):       #Lematyzuje zdania/ciągi słów i zmienia na
małe litery
```

```
    result=[]
    for word in words:
        word=lemmatizer.lemmatize(word,checkWordType(word))
        result.append(word)
    return result

def removeStops(words):     #Usuwa stopy z listy słów (i psuje
kolejność)
    return list(set(words)-set(stopwords.words('english')))

def cleanup_words_to_string(strings): #Oczyszcza listę słów ze stopów
i lematyzuje je
    result=""
    for word in removeStops(lematize(strings)):
        result+=word+" "
    return result[0:-1]

def cleanup_words_to_list(string): #To co funkcja wyżej, ale zwraca
listę zamiast długiego sklejanego łańcucha
    return removeStops(lematize(string))

df2['overview'] = [cleanup_words_to_list(x) for x in df2['overview']]
```

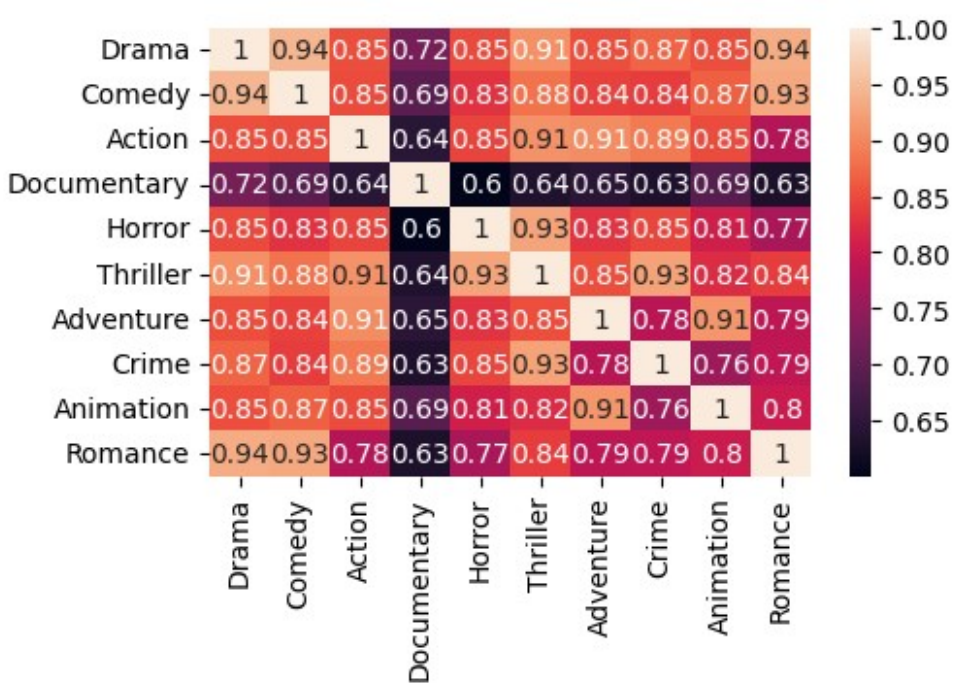
```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/azureuser/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/azureuser/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /home/azureuser/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /home/azureuser/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Po lematyzacji porównaliśmy słowa z wszystkich gatunków metryką kosinusową i wyznaczyliśmy macierz błędu podobieństwa, z której możemy odczytać najbardziej podobne do siebie gatunki.

```
def odleglosc(x, y, typ = 'kosinusowa'):
    if typ == 'kosinusowa':
        return np.dot(x, y) / (np.sqrt(np.dot(x, x)) *
np.sqrt(np.dot(y, y)))
    elif typ == 'Euklidesowa':
        return np.sqrt(np.sum((x - y) ** 2))
    else: # miejska
        return np.sum(np.abs(x - y))
```

```
genres_strings = OrderedDict.fromkeys(wanted_genres)
for key in genres_strings.keys():
    genres_strings[key] = ""

def make_combined_string_of_every_genre(row):
    for word in row[1]:
        genres_strings[row[0]] += " " + word

[make_combined_string_of_every_genre(x) for x in zip(df2['genres'],
df2['overview'])]
[make_combined_string_of_every_genre(x) for x in zip(df2['genres'],
df2['title'])]

genres_strings = list(genres_strings.values())

print("Liczby słów w haśle")
for i in range(0,len(wanted_genres)):
    print(wanted_genres[i]," -> ",len(wordlists[wanted_genres[i]]))

w = CountVectorizer()
s = np.array(w.fit_transform(genres_strings).todense())
s.shape

typ_odleglosci = "kosinusowa"
#typ_odleglosci = 'Euklidesowa'
mac_odl = np.empty([len(wanted_genres),len(wanted_genres)])
for i in range(0,len(wanted_genres)):
    for j in range(0,len(wanted_genres)):
        mac_odl[i,j] = odleglosc(s[i],s[j])
plt.figure(figsize=(5,3),dpi = 100)
sns.heatmap(pd.DataFrame(mac_odl, index = wanted_genres, columns =
wanted_genres),annot = mac_odl)
```

```
Liczby słów w haśle
Drama  ->  38022
Comedy  ->  32902
Action  ->  22473
Documentary  ->  20640
Horror  ->  14136
Thriller  ->  11453
Adventure  ->  11362
Crime  ->  10040
Animation  ->  10316
Romance  ->  9518

<matplotlib.axes._subplots.AxesSubplot at 0x7f0b2408ff10>
```



Z macierzy wynika, że dramat i komedia mają dużo podobnych słow, jednak zdecydowaliśmy się ich nie łączyć, ponieważ są to z zasady zupełnie inne gatunki, natomiast można również zauważyć, że horror, thriller i action mają dużo wspólnych słów, więc zdecydowaliśmy się połączyć te trzy gatunki w jeden.

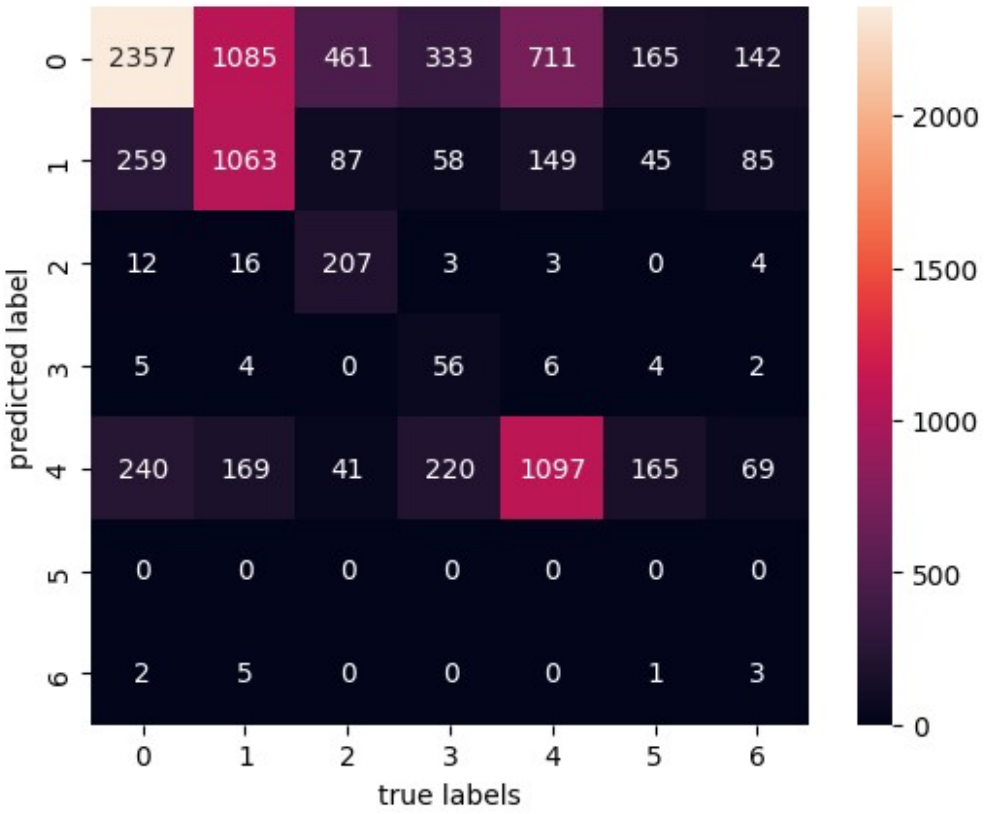
```
#funkcje pomocnicze
def merge_list_to_string(words):
    words_string = ""
    for word in words:
        words_string += " " + word
    return words_string

def build_training_text_data():
    test_data = []
    for element in df2['overview']:
        test_data.append(element)

    test_data = [merge_list_to_string(x) for x in test_data]

    test_target = []
    for element in df2['genres']:
        test_target.append(wanted_genres.index(element))

    return (test_data, test_target)

def classify_text(model, transform, test_data, test_target):
    #rozdział na dane uczące i testowe
    opis_ucz, opis_test, dec_ucz, dec_test =
train_test_split(test_data, test_target, test_size=0.3)
```

```
    #stworzenie modelu
    model = make_pipeline(transform, model)

    #uczenie modelu
    model.fit(opis_ucz, dec_ucz)

    #testowanie modelu
    predicted_categories = model.predict(opis_test)
    print(predicted_categories)
    test_score = model.score(opis_test, dec_test)
    learn_score = model.score(opis_ucz, dec_ucz)

    # macierz błędu
    mat = confusion_matrix(dec_test, predicted_categories)
    sns.heatmap(mat.T, square = True, annot=True, fmt = "d")
    plt.xlabel("true labels")
    plt.ylabel("predicted label")
    plt.show()

    return(learn_score, test_score)


def test_text_classification(model, transform, try_count, test_data,
test_target):
    learn_set_score = 0
    test_set_score = 0
    for k in range(0, try_count):
        data = split(df2, 0.3)
        learn_score, test_score = classify_text(model, transform,
test_data, test_target)
        learn_set_score += learn_score
        test_set_score += test_score
    print('Średnia dokładność z ' + str(try_count) + ' prób wynosi dla
zbioru uczącego ' + str(learn_set_score / try_count) + ' , a dla
zbioru testowego ' + str(test_set_score / try_count))

genres_to_merge = [['Thriller', 'Crime', 'Action']]
def merge_genres(x):
    for genres in genres_to_merge:
        if x in genres:
            return genres[0]
    return x

def remove_genres_from_list(genres_to_merge):
    for element in genres_to_merge:
        if(element!=genres_to_merge[0]) and (element in
wanted_genres):
            wanted_genres.remove(element)
```

```
df2['genres'] = [merge_genres(x) for x in df2['genres']]
remove_genres_from_list(genres_to_merge[0])

genres_to_merge = [['Comedy', 'Romance']]
df2['genres'] = [merge_genres(x) for x in df2['genres']]
remove_genres_from_list(genres_to_merge[0])

test_data=[]
test_target=[]


for element in df2['overview']:
    overview = ' '.join(element)
    test_data.append(overview)

for element in df2['genres']:
    test_target.append(wanted_genres.index(element))

opis_ucz, opis_test, dec_ucz, dec_test = train_test_split(test_data,
test_target, test_size=0.3)
#combined_classifier = VotingClassifier([('kneighbors',
KNeighborsClassifier(40)), ('SVM', SVC())], voting = 'soft', weights =
[0.4, 0.58])
model = make_pipeline(TfidfVectorizer(lowercase=True,
stop_words='english'), MultinomialNB(fit_prior=False))
#model = make_pipeline(TfidfVectorizer(lowercase=True,
stop_words='english'), combined_classifier)


model.fit(opis_ucz, dec_ucz)

print(wanted_genres)
predicted_categories = model.predict(opis_test)
print(predicted_categories)
prediction = model.score(opis_test, dec_test)
print(prediction)

# plot the confusion matrix
dec_test_names = []
for i in range(len(dec_test)):
    dec_test_names.append(wanted_genres[dec_test[i]])

predicted_categories_names = []
for i in range(len(predicted_categories)):

predicted_categories_names.append(wanted_genres[predicted_categories[i
]])

mat = confusion_matrix(dec_test, predicted_categories)
```

```
sns.heatmap(mat.T, square = True, annot=True, fmt = "d")
plt.xlabel("true labels")
plt.ylabel("predicted label")
plt.show()
```

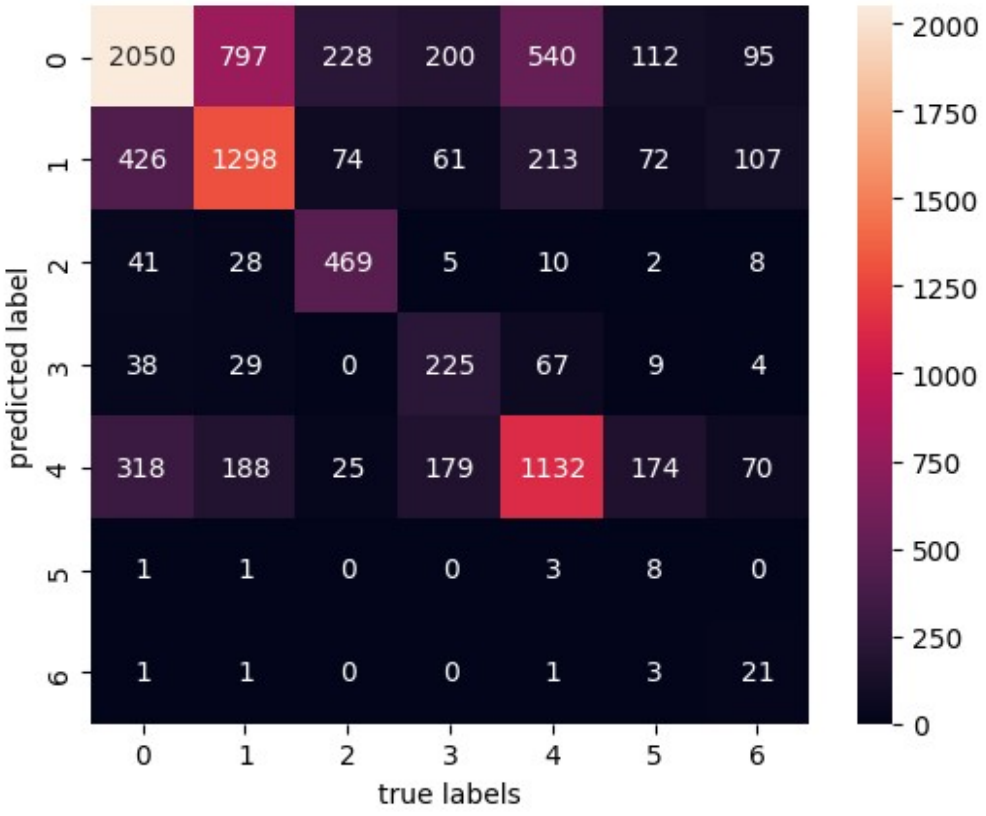
```
['Drama', 'Comedy', 'Documentary', 'Horror', 'Thriller', 'Adventure',
'Animation']
[2 0 0 ... 4 4 4]
0.512427683736876
[4]
```




```
model = make_pipeline(TfidfVectorizer(lowercase=True,
stop_words='english'), SVC())

model.fit(opis_ucz, dec_ucz)

predicted_categories = model.predict(opis_test)
print(predicted_categories)
prediction = model.score(opis_test, dec_test)
prediction2 = model.predict(['man tries to kidnap a woman in a park'])

print(prediction)
print(prediction2)

# plot the confusion matrix
```

```
dec_test_names = []
for i in range(len(dec_test)):
    dec_test_names.append(wanted_genres[dec_test[i]])

predicted_categories_names = []
for i in range(len(predicted_categories)):

predicted_categories_names.append(wanted_genres[predicted_categories[i
]])

mat = confusion_matrix(dec_test, predicted_categories)
sns.heatmap(mat.T, square = True, annot=True, fmt = "d")
plt.xlabel("true labels")
plt.ylabel("predicted label")
plt.show()
```

```
[2 0 0 ... 4 4 4]
0.5574244696807371
[4]
```




```
prediction = model.predict(['man tries to kidnap a woman in a park'])
print('Predykcja dla thrillera:', wanted_genres[prediction[0]])
prediction = model.predict(['a lot of blood and scary monsters'])
print('Predykcja dla horroru:', wanted_genres[prediction[0]])
prediction = model.predict(['tragic life of man and a woman'])
```

```
print('Predykcja dla dramatu:', wanted_genres[prediction[0]])
prediction = model.predict(['movie full of laughs and funny moments'])

print('Predykcja dla komedii:', wanted_genres[prediction[0]])
prediction = model.predict(['story of a great music composer, based on
a true story'])
print('Predykcja dla dokumentu:', wanted_genres[prediction[0]])
prediction = model.predict(['story full of adventures and fantasies'])

print('Predykcja dla filmu przygodowego:',
wanted_genres[prediction[0]])
prediction = model.predict(['an animation of your favourite cartoon
characters'])
print('Predykcja dla filmu animowanego:',
wanted_genres[prediction[0]])
```

```
Predykcja dla thrillera: Thriller
Predykcja dla horroru: Horror
Predykcja dla dramatu: Drama
Predykcja dla komedii: Comedy
Predykcja dla dokumentu: Drama
Predykcja dla filmu przygodowego: Drama
Predykcja dla filmu animowanego: Comedy
```

Wnioski: -Zbiory danych, które są dostępne bywają wybrakowane i trzeba dużo pracy włożyć w przefiltrowanie tych danych i przerobienie ich do postaci nadającej się do uczenia maszynowego. -Z klasyfikacji za pomocą naiwnego Bayesa widzimy, że najczęściej zgadzającymi się z rzeczywistością gatunkami są dramat, komedia i thriller, są to też gatunki, których było najwięcej w danych, a zarazem było przez to najwięcej różnych i unikalnych słów w tych gatunkach. Z testów wynika, że prawdopodobieństwo odgadnięcia gatunku wynosi około 50%. Nie jest to duże prawdopodobieństwo, ale jest znacząco lepsze od losowania gatunków (około 14% prawdopodobieństwa). Przy poprzednich próbach klasyfikacji prawdopodobieństwa były znaczenie niższe, i przerabianie danych klasyfikujących (takie jak lematyzacja, czy scalanie gatunków) powiększyło trafność klasyfikacji. Dla porównania użyliśmy też klasyfikatora SVC (również używanego do uczenia na bazie tekstu) i dzięki niemu osiągnęliśmy ponad 55% trafności, ale z drugiej strony uczenie i klasyfikacja ze pomocą tego klasyfikatora była czasochłonna (7 minut), w porównaniu do klasyfikacji Bayesem (kilka sekund). Daliśmy klasyfikatorom typowe opisy dla gatunków filmów do odgadnięcia i oba klasyfikatory dobrze poradziły sobie z thrillerem, horrorem, dramatem i komedią, a sam klasyfikator Bayesa dobrze odgadł jeszcze animację. Niestety dokumentu i filmu przegodowego nie udało się odganąć, pomimo dosyć oczywistych opisów. Podsumowując, nie jest to model, którego możnaby używać na codzień do odgadywania gatunku filmu, ponieważ na pewno są do tego lepsze narzędzia w internecie, jednak pokazuje on, że nierówna ilość informacji na temat danych może zaburzać klasyfikację i na korzyść nadmiarowych danych. -Język Python wymaga bardzo uważnego podejścia do typów danych, które nie są zdefiniowane w tym języku jawnie.