Open in Colab

In [ ]:
```
# INSTALL DEPENDENCIES
# Uncomment and run only once.
%pip install matplotlib numpy pandas scikit-learn scipy tensorflow pyclustering
```

```
Requirement already satisfied: matplotlib in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packa
ges (3.7.1)
Requirement already satisfied: numpy in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages
(1.23.5)
Requirement already satisfied: pandas in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages
(1.5.3)
Requirement already satisfied: scikit-learn in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pac
kages (1.2.2)
Requirement already satisfied: scipy in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages
(1.10.1)
Requirement already satisfied: tensorflow in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packa
ges (2.12.0rc1)
Requirement already satisfied: pyclustering in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pac
kages (0.10.1.2)
Requirement already satisfied: contourpy>=1.0.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site
-packages (from matplotlib) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pac
kages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/sit
e-packages (from matplotlib) (4.39.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/sit
e-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-
packages (from matplotlib) (23.0)
Requirement already satisfied: pillow>=6.2.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pa
ckages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site
-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/
site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pac
kages (from pandas) (2022.7.1)
Requirement already satisfied: joblib>=1.1.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pa
ckages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/
site-packages (from scikit-learn) (3.1.0)
Requirement already satisfied: absl-py>=1.0.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-p
ackages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/sit
e-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site
-packages (from tensorflow) (23.3.3)
```

```
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/s
ite-packages (from tensorflow) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/s
ite-packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/s
ite-packages (from tensorflow) (1.51.3)
Requirement already satisfied: h5py>=2.9.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pack
ages (from tensorflow) (3.8.0)
Requirement already satisfied: jax>=0.3.15 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pack
ages (from tensorflow) (0.4.5)
Requirement already satisfied: keras<2.13,>=2.12.0rc0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.1
1/site-packages (from tensorflow) (2.12.0rc1)
Requirement already satisfied: libclang>=13.0.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site
-packages (from tensorflow) (15.0.6.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/sit
e-packages (from tensorflow) (3.3.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=
3.20.3 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from tensorflow) (4.22.1)
Requirement already satisfied: setuptools in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packa
ges (from tensorflow) (65.5.0)
Requirement already satisfied: six>=1.12.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-pack
ages (from tensorflow) (1.16.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.
11/site-packages (from tensorflow) (2.12.0)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0rc0 in /home/thanos/.pyenv/versions/3.11.
2/lib/python3.11/site-packages (from tensorflow) (2.12.0rc0)
Requirement already satisfied: termcolor>=1.1.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site
-packages (from tensorflow) (2.2.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /home/thanos/.pyenv/versions/3.11.2/lib/python
3.11/site-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/s
ite-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /home/thanos/.pyenv/versions/3.11.2
/lib/python3.11/site-packages (from tensorflow) (0.31.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/si
te-packages (from astunparse>=1.6.0->tensorflow) (0.38.4)
Requirement already satisfied: google-auth<3,>=1.6.3 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11
/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (2.16.2)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /home/thanos/.pyenv/versions/3.11.2/lib
/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (0.4.6)
Requirement already satisfied: markdown>=2.6.8 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-
packages (from tensorboard<2.13,>=2.12->tensorflow) (3.4.1)
```

```
Requirement already satisfied: requests<3,>=2.21.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (2.28.2)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (0.7.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (1.8.1)
Requirement already satisfied: werkzeug>=1.0.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from tensorboard<2.13,>=2.12->tensorflow) (2.2.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow) (5.3.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.13,>=2.12->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow) (3.1.0)
Requirement already satisfied: idna<4,>=2.5 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow) (1.26.14)
Requirement already satisfied: certifi>=2017.4.17 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow) (2022.12.7)
Requirement already satisfied: MarkupSafe>=2.1.1 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from werkzeug>=1.0.1->tensorboard<2.13,>=2.12->tensorflow) (2.1.2)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.13,>=2.12->tensorflow) (3.2.2)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
# IMPORTS AND GLOBAL CONSTANTS

# Load the TensorBoard notebook extension
%load_ext tensorboard

import math
import tensorflow as tf
import datetime, os
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import typing
import numpy.typing as np_typing
from sklearn.model_selection import train_test_split

##MAIN PROGRAM VARIABLES##

# Constants
DATASET_FILE_PATH = "./Dataset.npy"
#Define the figures path
# FIGURES_PATH = "figures"
# os.makedirs(FIGURES_PATH, exist_ok=True)
# #Define the data folder path
DATAFOLDER_PATH = "datafiles"
os.makedirs(DATAFOLDER_PATH, exist_ok=True)
RESULTS_PATH = "results"
os.makedirs(RESULTS_PATH, exist_ok=True)

L_CLUSTERS_NUM = 5
K_NEIGHBORS_NUM = 6

if 'google.colab' in str(get_ipython()):
  print('Running on CoLab')
  from google.colab import drive
  drive.mount('/content/drive/')
  DATASET_FILE_PATH = "/content/drive/My Drive/Colab Notebooks/Dataset.npy"
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [ ]:
```python
def calculate_ratings_matrix_df_W_CommonRatings():
    dataset: np.ndarray = np.load(DATASET_FILE_PATH)
    display('dataset.size', dataset.size)

    #### CELL 1
    #Define the splitter lambda function in order to tokenize the initial string data.
    splitter = lambda s: s.split(",")
    #Apply the splitter lambda function on the string np array
    dataset = np.array([splitter(x) for x in dataset])
    #Set the pickle file for storing the initial dataframe
    pickle_file = os.path.join(DATAFOLDER_PATH, "dataframe.pkl")
    #Check the existence of the specified file.
    if os.path.exists(pickle_file):
        #Load the pickle file
        dataframe = pd.read_pickle(pickle_file)
    else:
        #Create the dataframe object.
        dataframe = pd.DataFrame(dataset, columns=['User','Movie','Rating','Date'])
        #Convert the string elements of the "Users" series into integers
        dataframe["User"] = dataframe["User"].apply(lambda s:np.int64(s.replace("ur","")))
        #Convert the string elements of the "Movies" series into integers
        dataframe["Movie"] = dataframe["Movie"].apply(lambda s:np.int64(s.replace("tt","")))
        #Convert the string elements of the "Ratings" series into integers
        dataframe["Rating"] = dataframe["Rating"].apply(lambda s:np.int64(s))
        #Convert the string element of "Dates" series into datetime Object
        dataframe["Date"] = pd.to_datetime(dataframe["Date"])
        dataframe.to_pickle(pickle_file)


    #### CELL 2
    #Get the unique users in the dataset.
    users = dataframe["User"].unique()
    #Get the number of unique users
    users_num = len(users)
    #Get the unique movie items in the dataset.
    movies = dataframe["Movie"].unique()
    #Get the number of unique movies
    movies_num = len(movies)
    #Get the total number of existing ratings.
    ratings_num = dataframe.shape[0]
    #Report the number of unique Users and Movies in the dataset
    display("INITIAL DATASET: {0} number of unique users and {1} of unique movies".format(users_num, movies
```

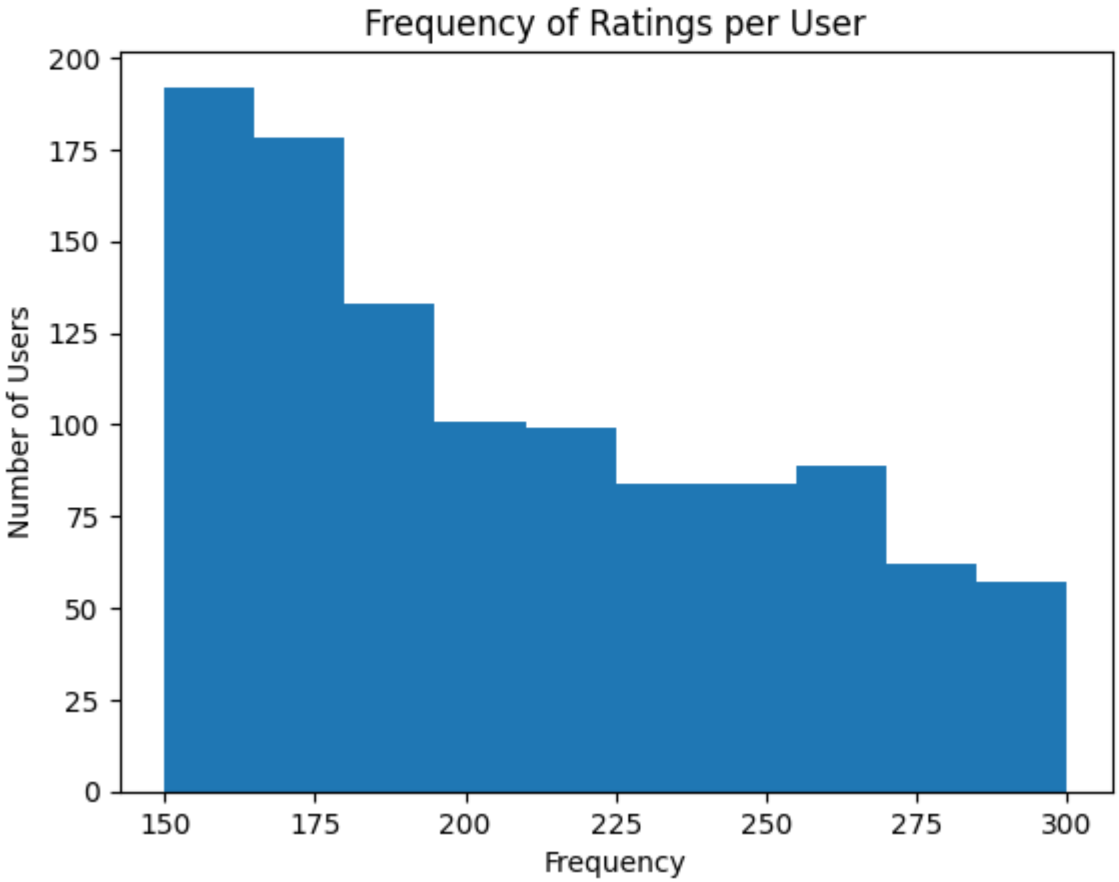
```
display("INITIAL DATASET: {} number of unique users and {} of unique movies".format(users_num, movies
#Report the total number of existing ratings in the dataset
display("INITIAL DATASET: {} total number of existing ratings".format(ratings_num))



# CELL 3
#Define the pickle file that will store the time span per user dataframe
pickle_file = os.path.join(DATAFOLDER_PATH, "ratings_num_df.pkl")
#Check the existence of the previously defined pickle file
if os.path.exists(pickle_file):
    #Load the pickle file
    ratings_num_df = pd.read_pickle(pickle_file)
else:
    ratings_num_df = dataframe.groupby("User")["Rating"].count().sort_values(ascending=False).reset_inc
    #Save the previously created dataframe to pickle
    ratings_num_df.to_pickle(pickle_file)



# CELL 4
#Set the pickle file that will store the time span per user dataframe
pickle_file = os.path.join(DATAFOLDER_PATH, "ratings_span_df.pkl")
if os.path.exists(pickle_file):
    ratings_span_df = pd.read_pickle(pickle_file)
else:
    ratings_span_df = dataframe.groupby("User")["Date"].apply(lambda date: max(date)-min(date)).sort_va
    ratings_span_df.to_pickle(pickle_file)
#Create a new ratings dataframe by joining the previously defined dataframe
ratings_df = ratings_num_df.join(ratings_span_df.set_index("User"),on="User")
ratings_df["ratings_span"]=ratings_df["ratings_span"].dt.days
#Set the threshold values for the minimum and maximum number of Ratings per user
minimum_ratings = 150
maximum_ratings = 300
#Discard all users that do not pertain to the previous range of ratings
reduced_ratings_df = ratings_df.loc[(ratings_df["ratings_num"] >= minimum_ratings) & (ratings_df["ratir

#Generate the frequency histogram for the number of ratings per user
reduced_ratings_df["ratings_num"].plot(kind='hist', title='Frequency of Ratings per User', xticks=range
plt.xlabel('Frequency')
plt.ylabel('Number of Users')

plt.show()
```

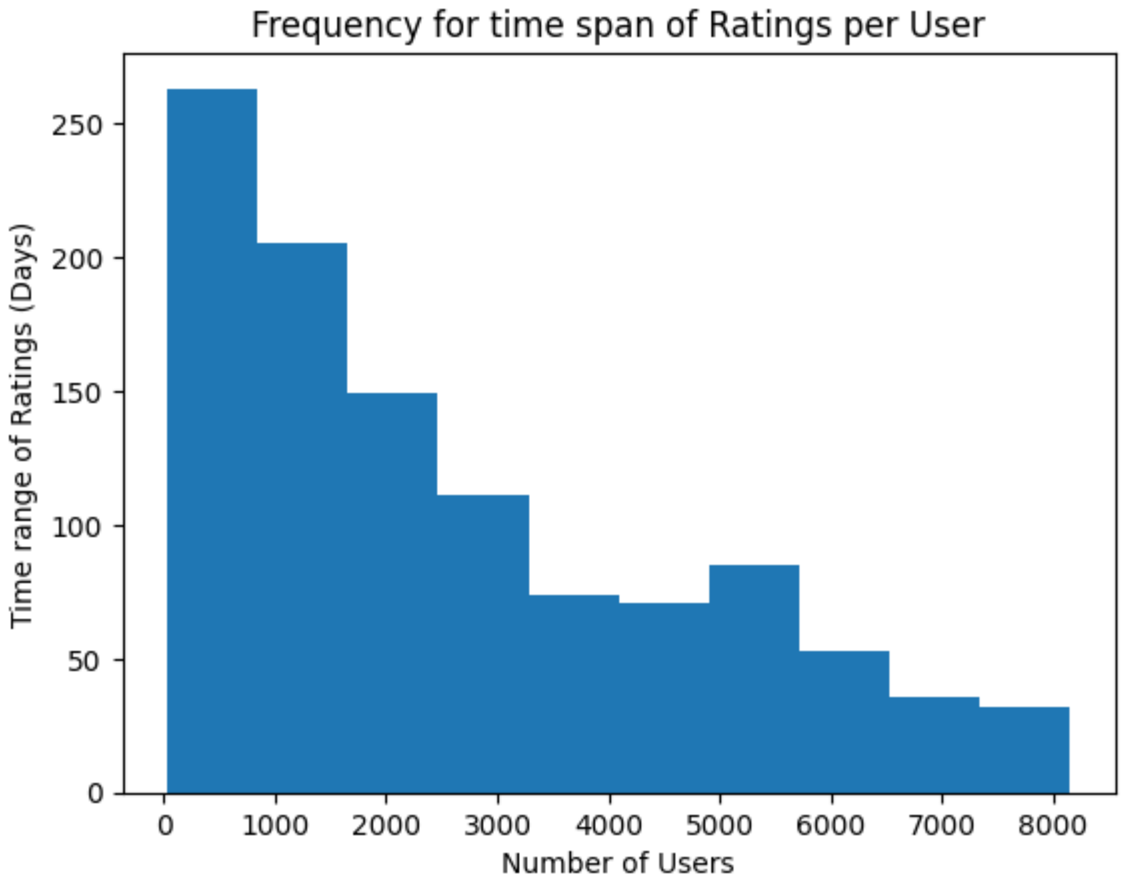
```
#Generate the frequency histogram for the time span of ratings per user
reduced_ratings_df["ratings_span"].plot(kind='hist', title='Frequency for time span of Ratings per User
plt.xlabel('Number of Users')
plt.ylabel('Time range of Ratings (Days)')

plt.show()



#### CELL 5
#Get the final dataframe by excluding all users whose ratings fall outside the prespecified range
final_df = dataframe.loc[dataframe["User"].isin(reduced_ratings_df["User"])].reset_index()
#Drop the links (indices) to the original table
final_df = final_df.drop("index", axis=1)
#Get the unique users and items in the final dataframe along with the final number of ratings
final_users = final_df["User"].unique()
final_movies = final_df["Movie"].unique()
final_users_num = len(final_users)
final_movies_num = len(final_movies)
final_ratings_num = len(final_df)

#Report the final number of unique users and movies in the dataset
display("REDUCED DATASET: {0} number of unique users and {1} number of unique movies".format(final_user
#Report the final number of existing ratings in the dataset
display("REDUCED DATASET: {} number of existing ratings in the dataset".format(final_ratings_num))



# CELL 6
#We need to reset the users and items IDs in order to be able to construct a network of users and Movie
#Users and Movies IDs should be consecutive in the [1..final_users_num] and [1...final_movies_num]
#Initially, we need to acquire the sorted versions of the user and movies
sorted_final_users = np.sort(final_users)
sorted_final_movies = np.sort(final_movies)
#Generate the dictionary of final users as a mapping of the following
#sorted_final_users --> [0...final_users_num-1]
final_users_dict = dict(zip(sorted_final_users,list(range(0,final_users_num))))
#Generate the dictionary of final items as a mapping of the following
final_movies_dict = dict(zip(sorted_final_movies,list(range(0,final_movies_num))))
#Apply the previously defined dictionary-based maps on the users and movies columns of the final dataf
final_df["User"] = final_df["User"].map(final_users_dict)
final_df["Movie"] = final_df["Movie"].map(final_movies_dict)
```

```
#Get a grouped version of the original dataframe based on the unique final users
users_group_df = final_df.groupby("User")
#Initialize the adjacency matrix which stores the connection status for pair of users in the recommenda
W = np.zeros((final_users_num, final_users_num))
#Iinitialize the matrix storing the number of commonly rated items for a pair of users
CommonRatings = np.zeros((final_users_num, final_users_num))
#Initialize the matrix of common ratings
#Matrix W will be of size [final_users_num x final_users_num],
#Let U = {u1, u2,...,un} be the final set of users and I = {i1,i2,...,im}
#final set of movies. By considering the function Fi: U -> P(I) where
#P(I) is the powerset of I, Fi(u) returns the subset of items that has been rated by user u.
#In this context, the edge weight between any given pair of users (u,v) will be computed as:
#
#          |Intersection(Fi(u)),Fi(v))|
#W(u,v) =  -----------------------------
#               |Union(Fi(u),Fi(v))|
#
#
#In order to speed up the construction of the adjacency matrix for the ratings network,
#construct a dictionary object that will store a set of rated items for each unique user.
user_items_dict = {}
# for user in final_users:
    #print(user)
    # user_index = final_users_dict[user]
    # user_movies = set(users_group_df.get_group(user_index)["Movie"])
    # user_items_dict[user_index] = user_movies

# Initialize the dictionary for storing the set of rated items for each user
user_items_dict = {}
# print(final_users_dict)
# print(sorted_final_users)
# print(final_users_dict)
# For each unique user, find the set of movies that they rated
for user in final_users:
    if user in final_users_dict:
        user_index = final_users_dict[user]
        user_movies = set(users_group_df.get_group(user_index)["Movie"])
        user_items_dict[user_index] = user_movies



#### CELL 7
user_ids = list(user_items_dict.keys())
```

```
    user_ids.sort()
    #Generate the sorted version of the dictionary
    user_items_dict = {user_index:user_items_dict[user_index] for user_index in user_ids}
    #Set the pickle file that will store the graph adjacency matrix W.
    pickle_file_weights = os.path.join(DATAFOLDER_PATH, "w.npy")
    pickle_file_common_ratings = os.path.join(DATAFOLDER_PATH, "common_ratings.npy")
    #Check the existence of the previously defined pickle file
    if os.path.exists(pickle_file_weights) & os.path.exists(pickle_file_common_ratings):
        #Load the pickle file
        W = np.load(pickle_file_weights)
        CommonRatings = np.load(pickle_file_common_ratings)
    else:
        for source_user in user_items_dict.keys():
            for target_user in user_items_dict.keys():
                intersection_items = user_items_dict[source_user].intersection(user_items_dict[target_user]
                union_items = user_items_dict[source_user].union(user_items_dict[target_user])
                W[source_user, target_user] = len(intersection_items)/len(union_items)
                CommonRatings[source_user, target_user] = len(intersection_items)
        np.save(pickle_file_weights,W)
        np.save(pickle_file_common_ratings,CommonRatings)



    # Create a pivot table of user-movie ratings
    ratings_matrix_df = final_df.pivot_table(index='User', columns='Movie', values='Rating')
    ratings_matrix_df = ratings_matrix_df.fillna(0)

    ratings_matrix_array = ratings_matrix_df.to_numpy()

    display('ratings_matrix_df', ratings_matrix_df)
    display('ratings_matrix_array', ratings_matrix_array)

    #### OUTPUT
    return ratings_matrix_df, W, CommonRatings


ratings_matrix_df, W, CommonRatings = calculate_ratings_matrix_df_W_CommonRatings()
ratings_matrix_array = ratings_matrix_df.to_numpy()
display('W', W)
display('CommonRatings', CommonRatings)
display('ratings_matrix_df', ratings_matrix_df)
display('ratings_matrix_array', ratings_matrix_array)
```

```
'dataset.size'
4669820
'INITIAL DATASET: 1499238 number of unique users and 351109 of unique movies'
'INITIAL DATASET: 4669820 total number of existing ratings'
```






```
'REDUCED DATASET: 1079 number of unique users and 68084 number of unique movies'
'REDUCED DATASET: 224704 number of existing ratings in the dataset'
'ratings_matrix_df'
```

| Movie | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 68074 | 68075 | 68076 | 68077 | 68078 | 68079 | 68080 | 68081 | 68082 | 68083 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **User** | | | | | | | | | | | | | | | | | | | | | |
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1074** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1075** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1076** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| **1077** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1078** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

1079 rows × 68084 columns

```
'ratings_matrix_array'
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
'W'
```

```
array([[1.        , 0.01627907, 0.00228311, ..., 0.00228311, 0.0017331 ,
        0.00998004],
       [0.01627907, 1.        , 0.00657895, ..., 0.00657895, 0.        ,
        0.        ],
       [0.00228311, 0.00657895, 1.        , ..., 0.        , 0.01360544,
        0.        ],
       ...,
       [0.00228311, 0.00657895, 0.        , ..., 1.        , 0.00224215,
        0.0026738 ],
       [0.0017331 , 0.        , 0.01360544, ..., 0.00224215, 1.        ,
        0.        ],
       [0.00998004, 0.        , 0.        , ..., 0.0026738 , 0.        ,
        1.        ]])
'CommonRatings'
array([[285.,   7.,   1., ...,   1.,   1.,   5.],
       [  7., 152.,   2., ...,   2.,   0.,   0.],
       [  1.,   2., 154., ...,   0.,   6.,   0.],
       ...,
       [  1.,   2.,   0., ..., 154.,   1.,   1.],
       [  1.,   0.,   6., ...,   1., 293.,   0.],
       [  5.,   0.,   0., ...,   1.,   0., 221.]])
'ratings_matrix_df'
```

| Movie | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 68074 | 68075 | 68076 | 68077 | 68078 | 68079 | 68080 | 68081 | 68082 | 68083 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **User** | | | | | | | | | | | | | | | | | | | | | |
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1074** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1075** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1076** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| **1077** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1078** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

1079 rows × 68084 columns

```
'ratings_matrix_array'
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

# Δημιουργούμε έναν πίνακα χρηστών - ταινιών

(οι χρήστες βρίσκονται στις γραμμές και οι ταινίες στις στήλες του πίνακα) όπου τα στοιχεία του πίνακα είναι από 1 - 10. Εάν ο χρήστης δεν έχει αξιολογήσει την ταινία, η αξιολόγηση που θα ανατεθεί είναι 0.

In [ ]:
```
from pyclustering.cluster.kmeans import kmeans, kmeans_visualizer
from pyclustering.cluster.center_initializer import kmeans_plusplus_initializer
from pyclustering.samples.definitions import FCPS_SAMPLES
from pyclustering.utils import read_sample
from pyclustering.cluster.kmeans import kmeans
from pyclustering.utils.metric import type_metric, distance_metric
```

Θέλουμε να δημιουργήσουμε τον πίνακα βαρών "λ" των χρηστών. Τον πίνακα αξιολογήσεων δηλαδή όπου η τιμή της αξιολόγησης είναι 1 εάν η ταινία έχει αξιολογηθεί από τον χρήστη ή 0 εάν δεν έχει αξιολογηθεί

In [ ]:
```
# Threshold
threshold = 1

# Transform to binary
binary_matrix = np.where(ratings_matrix_df >= threshold, 1, 0)
display('binary_matrix', binary_matrix)
```

```
'binary_matrix'
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [ ]:
```
# Convert the matrix to a numpy array

# Create a dictionary that maps each row of the matrix to its index
# matrix_dict = {tuple(row): i for i, row in enumerate(matrix_array)}
```

# *Αλγόριθμοι Ομαδοποίησης Δεδομένων*

**Χρήση της Weighted Euclidean Distance**

In [ ]:
```
from scipy.spatial.distance import pdist, cdist
import numpy as np

from scipy.sparse import csr_matrix

def pairwise_weighted_euclidean_distance(X, weights):
    # Find the indices of the rated movies for each pair of users
    rated_movies = (weights_sparse.T @ weights_sparse) > 0

    # Select only the rated movies for each pair of users
    X_rated = X_sparse[:, rated_movies]

    # Calculate the pairwise weighted Euclidean distance between
    #users who have rated the same movie
    return cdist(X, metric='euclidean')

def kmeans_pairwise_weighted_euclidean(X, weights, k, max_iters=2):

    n, m = X.shape
    centroids = X[np.random.choice(n, k, replace=False)]
    distances = pairwise_weighted_euclidean_distance(X, weights)
    for i in range(max_iters):
        # Assign points to clusters
        cluster_assignments = np.argmin(distances, axis=1)

        # Recalculate cluster centroids
        for j in range(k):
            cluster_points = X[cluster_assignments == j]
            if len(cluster_points) > 0:
                centroids[j] = np.average(cluster_points, axis=0)

        # Update distances to centroids
        distances = pairwise_weighted_euclidean_distance(X, weights)

    return cluster_assignments, centroids
```

# Clustering users using K-means

We want to start by creating the symmetric D matrix which contains the pairwise weighted Euclidean distance for every pair of users. We calculate the distance between each user using

- dist_{u,v}=\sum_{k=1}^{n}\sqrt{|R_{u}(k) - R_{v}(k)|λ*{u}(k)λ*{v}(k)}

In [ ]:
```
# Calculate the pairwise weighted Euclidean distance matrix

def create_euclidean_distance_matrix_cached(ratings_matrix: pd.DataFrame, binary_matrix: np_typing.NDArray)
    #Set the npy file that will store the Euclidean distance matrix
    npy_file = os.path.join(DATAFOLDER_PATH, "euclidean_distance_matrix.npy")
    if os.path.exists(npy_file):
        Dist_euclidean: np_typing.NDArray[np.float64] = np.load(npy_file, allow_pickle=True)
        return Dist_euclidean
    else:
        n = ratings_matrix.shape[0]
        Dist_euclidean = np.zeros((n, n))
        for i in range(n):
            for j in range(i, n):
              if i == j:
                d = 0
              else:
                d = np.sqrt(np.sum(binary_matrix[i,:]*binary_matrix[j,:] * (ratings_matrix.iloc[i,:] - rat
                Dist_euclidean[i,j] = d
                Dist_euclidean[j,i] = d
        np.save(npy_file, Dist_euclidean, allow_pickle=True, fix_imports=True)
        return Dist_euclidean


Dist_euclidean = create_euclidean_distance_matrix_cached(ratings_matrix_df, binary_matrix)
Dist_euclidean
```

```
Out[ ]: array([[0.        , 3.46410162, 1.        , ..., 4.        , 4.        ,
          3.31662479],
         [3.46410162, 0.        , 7.07106781, ..., 5.        , 0.        ,
          0.        ],
         [1.        , 7.07106781, 0.        , ..., 0.        , 8.06225775,
          0.        ],
         ...,
         [4.        , 5.        , 0.        , ..., 0.        , 1.        ,
          0.        ],
         [4.        , 0.        , 8.06225775, ..., 1.        , 0.        ,
          0.        ],
         [3.31662479, 0.        , 0.        , ..., 0.        , 0.        ,
          0.        ]])
```

```
In [ ]: df_euclidean = pd.DataFrame(Dist_euclidean)
        df_euclidean
```

Out[ ]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1069 | 1070 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.000000 | 3.464102 | 1.000000 | 3.872983 | 9.000000 | 0.000000 | 7.141428 | 0.000000 | 4.00000 | 3.872983 | ... | 4.358899 | 0.0 | 10.4 |
| **1** | 3.464102 | 0.000000 | 7.071068 | 2.236068 | 2.449490 | 7.874008 | 6.000000 | 3.316625 | 1.00000 | 2.828427 | ... | 3.316625 | 0.0 | 9.4 |
| **2** | 1.000000 | 7.071068 | 0.000000 | 13.266499 | 6.164414 | 17.549929 | 6.082763 | 4.898979 | 1.00000 | 9.949874 | ... | 3.464102 | 0.0 | 5.6 |
| **3** | 3.872983 | 2.236068 | 13.266499 | 0.000000 | 10.099505 | 9.486833 | 4.898979 | 7.810250 | 4.00000 | 3.162278 | ... | 5.000000 | 0.0 | 0.0 |
| **4** | 9.000000 | 2.449490 | 6.164414 | 10.099505 | 0.000000 | 9.219544 | 5.830952 | 4.358899 | 2.44949 | 10.908712 | ... | 9.746794 | 0.0 | 9.1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **1074** | 0.000000 | 3.000000 | 4.242641 | 1.000000 | 0.000000 | 5.000000 | 1.000000 | 0.000000 | 0.00000 | 0.000000 | ... | 5.099020 | 0.0 | 8.7 |
| **1075** | 1.000000 | 2.828427 | 5.196152 | 9.000000 | 2.449490 | 9.433981 | 3.000000 | 4.242641 | 1.00000 | 8.774964 | ... | 7.280110 | 0.0 | 6.6 |
| **1076** | 4.000000 | 5.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.00000 | 3.000000 | ... | 7.280110 | 0.0 | 0.0 |
| **1077** | 4.000000 | 0.000000 | 8.062258 | 2.000000 | 6.324555 | 3.000000 | 0.000000 | 0.000000 | 0.00000 | 6.164414 | ... | 1.000000 | 0.0 | 7.3 |
| **1078** | 3.316625 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.44949 | 2.000000 | ... | 3.162278 | 0.0 | 9.8 |

1079 rows × 1079 columns

Στον πίνακα αποστάσεων που έχουμε δημιουργήσει, θα τρέξουμε τον αλγόριθμο k-means ώστε να αποτιμήσουμε την ομοιότητα των χρηστών χρησιμοποιώντας τις μεταξύ τους αποστάσεις.

In [ ]:
```
from sklearn.cluster import KMeans
# Cluster the users using K-means
kmeans = KMeans(n_clusters=L_CLUSTERS_NUM).fit(Dist_euclidean)

# Get the cluster labels
labels_euclidean = kmeans.labels_

# Print the labels
print(labels_euclidean)
```

```
/home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
[1 1 2 ... 1 0 0]
```

Cluster the users, by using a custom dist = 1 - np.abs(np.sum(R_uR_vweights_uweights_l)/(np.sqrt(R^2_uweights_uweights_l)np.sqrt(R^2_vweights_uweights_l)

---

In [ ]:
```python
# Calculate the pairwise weighted Cosine distance matrix

def create_cosine_distance_matrix_cached(ratings_matrix: pd.DataFrame, binary_matrix: np_typing.NDArray) ->
    #Set the npy file that will store the Euclidean distance matrix
    npy_file = os.path.join(DATAFOLDER_PATH, "cosine_distance_matrix.npy")
    if os.path.exists(npy_file):
        Dist_cosine: np_typing.NDArray[np.float64] = np.load(npy_file, allow_pickle=True)
        return Dist_cosine
    else:
        n = ratings_matrix.shape[0]
        Dist_cosine = np.zeros((n, n))
        for i in range(n):
            for j in range(i, n):
              if i == j :
                d = 0
              else:
                d = 1 - np.abs(np.sum(binary_matrix[i,:] * binary_matrix[j,:] * ratings_matrix.loc[i,:] * r
                Dist_cosine[i,j] = d
                Dist_cosine[j,i] = d
        np.save(npy_file, Dist_cosine, allow_pickle=True, fix_imports=True)
        return Dist_cosine


Dist_cosine = create_cosine_distance_matrix_cached(ratings_matrix_df, binary_matrix)
Dist_cosine
```

Out[ ]:
```
array([[ 0.        , -3.8226601 , -1.44948974, ..., -1.44948974,
        -0.41421356, -3.06815636],
       [-3.8226601 ,  0.        , -1.34980346, ..., -1.21988247,
                nan,         nan],
       [-1.44948974, -1.34980346,  0.        , ...,         nan,
        -3.10991495,         nan],
       ...,
       [-1.44948974, -1.21988247,         nan, ...,  0.        ,
        -1.64575131, -2.16227766],
       [-0.41421356,         nan, -3.10991495, ..., -1.64575131,
         0.        ,         nan],
       [-3.06815636,         nan,         nan, ..., -2.16227766,
                nan,  0.        ]])
```

In [ ]:
```
df_cosine = pd.DataFrame(Dist_cosine)
df_cosine = df_cosine.replace(np.nan, 0)
df_cosine
```

Out[ ]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1069 | 1070 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.000000 | -3.822660 | -1.449490 | -2.881451 | -2.272729 | 0.000000 | -2.529866 | 0.000000 | -2.000000 | -3.012037 | ... | -4.361581 | 0.0 | -1 |
| **1** | -3.822660 | 0.000000 | -1.349803 | -2.276504 | -3.322703 | -1.718926 | -3.300121 | -2.655695 | -1.828427 | -1.793871 | ... | -3.314440 | 0.0 | -( |
| **2** | -1.449490 | -1.349803 | 0.000000 | -2.736446 | -3.337589 | -0.667852 | -1.661013 | -3.971771 | -2.661735 | -1.602330 | ... | -4.073196 | 0.0 | -3 |
| **3** | -2.881451 | -2.276504 | -2.736446 | 0.000000 | -3.412095 | -0.664242 | -1.898415 | -2.821065 | -1.828427 | -0.778279 | ... | -2.577866 | 0.0 | -1 |
| **4** | -2.272729 | -3.322703 | -3.337589 | -3.412095 | 0.000000 | -1.216870 | -3.383925 | -3.643170 | -3.689046 | -2.812883 | ... | -4.162197 | 0.0 | -2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **1074** | 0.000000 | -1.236068 | -1.912951 | -1.645751 | 0.000000 | -1.449490 | -1.236068 | 0.000000 | 0.000000 | -1.236068 | ... | -1.952385 | 0.0 | -3 |
| **1075** | -2.464112 | -3.453007 | -3.367817 | -3.340152 | -3.236119 | -2.578231 | -1.731552 | -3.034495 | -1.645751 | -2.919914 | ... | -3.933937 | 0.0 | -3 |
| **1076** | -1.449490 | -1.219882 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -1.000000 | -2.210380 | ... | -2.341791 | 0.0 | ( |
| **1077** | -0.414214 | 0.000000 | -3.109915 | -1.000000 | -1.696309 | -1.000000 | 0.000000 | 0.000000 | 0.000000 | -2.519201 | ... | -2.871324 | 0.0 | -1 |
| **1078** | -3.068156 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -2.606767 | -1.828427 | ... | -3.949221 | 0.0 | -1 |

1079 rows × 1079 columns

Στον πίνακα αποστάσεων που έχουμε δημιουργήσει, θα τρέξουμε τον αλγόριθμο k-means ώστε να αποτιμήσουμε την ομοιότητα των χρηστών χρησιμοποιώντας τις μεταξύ τους αποστάσεις.

In [ ]:
```
# Cluster the users using K-means
kmeans = KMeans(n_clusters=L_CLUSTERS_NUM).fit(df_cosine)

# Get the cluster labels
labels_cosine = kmeans.labels_

# Print the labels
print(labels_cosine)
```

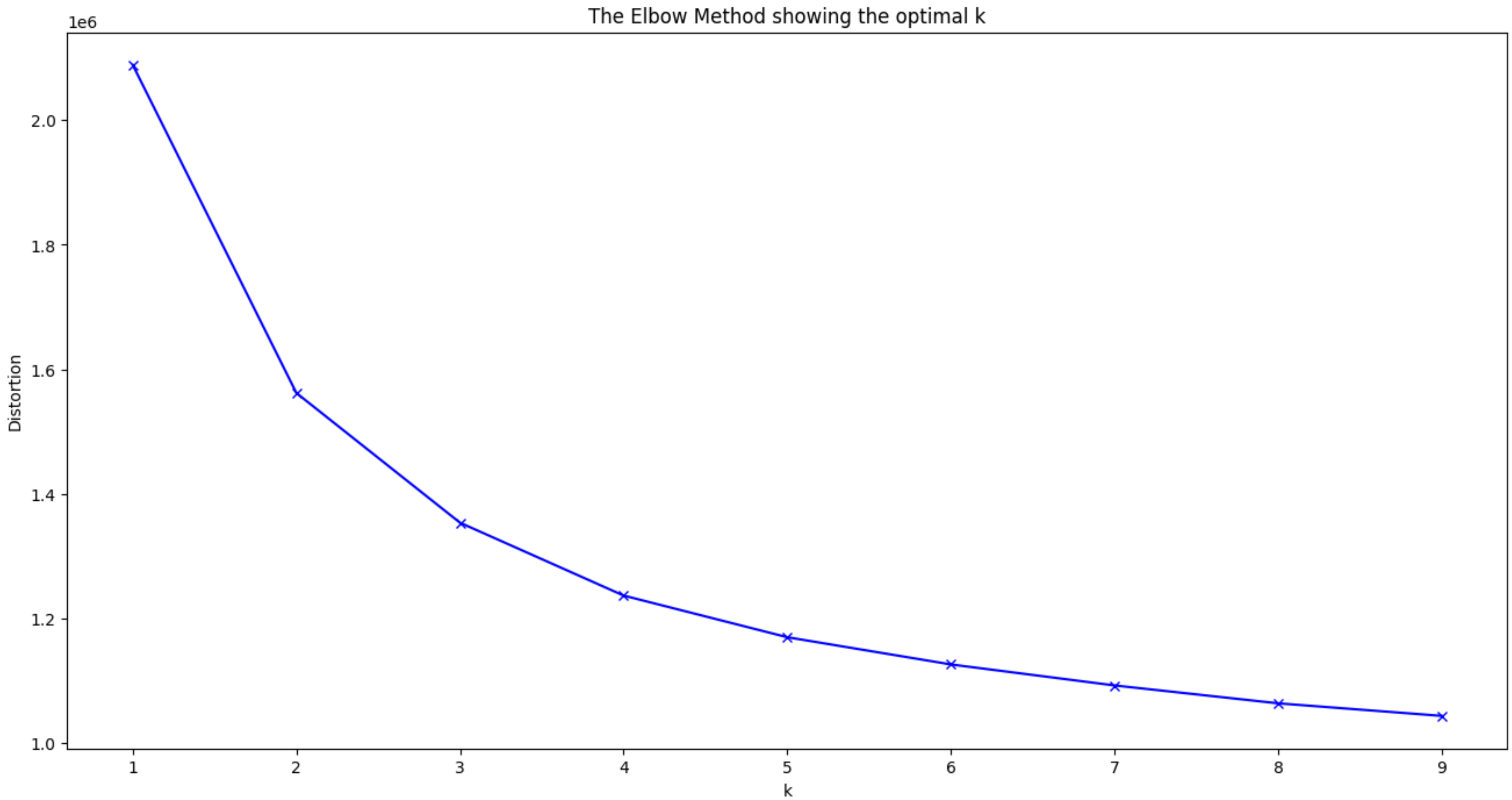
```
/home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explic
itly to suppress the warning
  warnings.warn(
[4 3 3 ... 1 3 3]
```

# Elbow Method

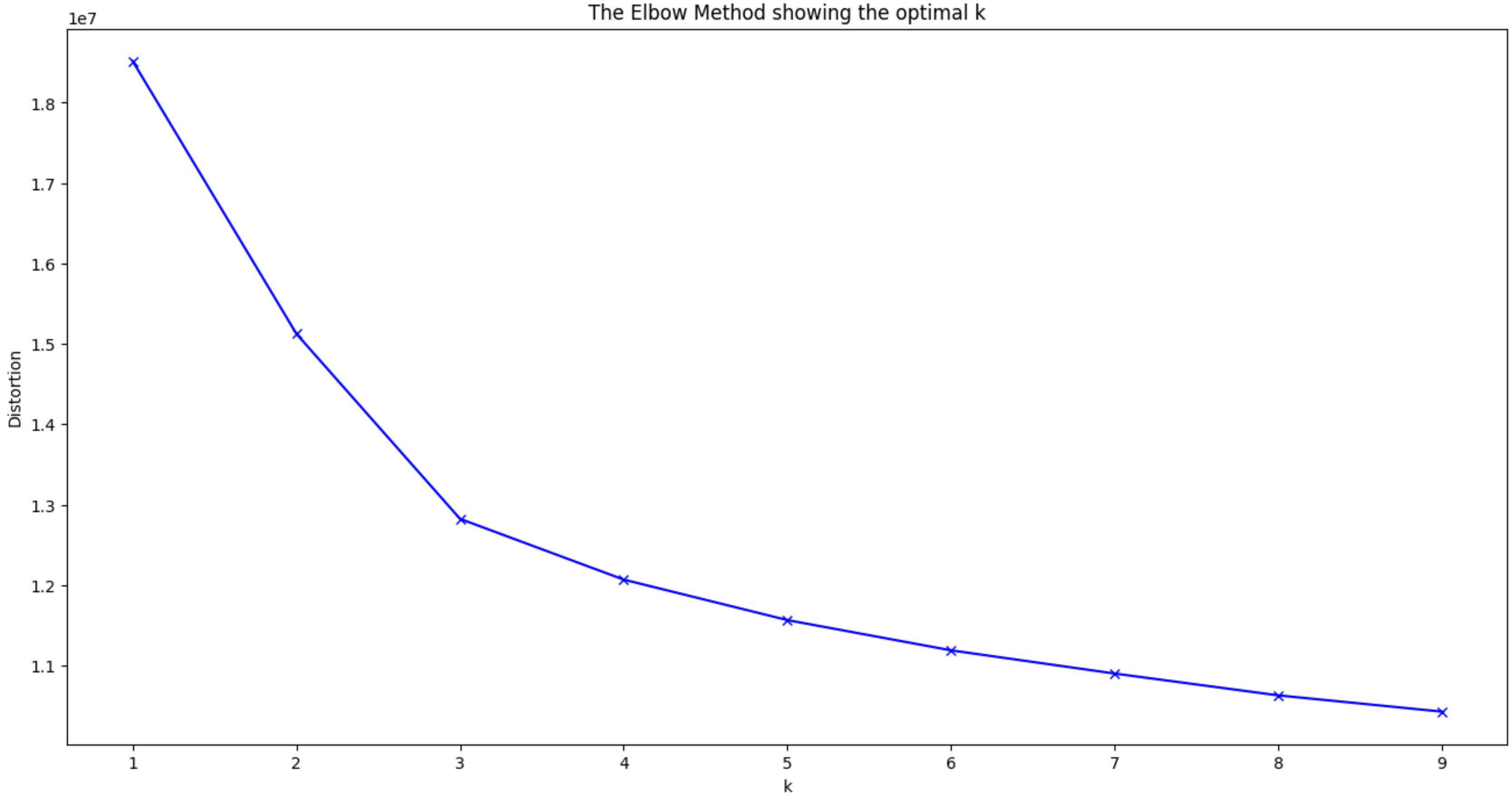
Χρησιμοποιούμε την elbow method ώστε να επιλέξουμε τον βέλτιστο αριθμό clusters στον οποίο θα διαχωριστούν τα δεδομένα χρησιμοποιώντας τον k-means

In [ ]:
```
def elbow_method(df: pd.DataFrame, max_iter: int):
  distortions = []
  K = range(1,max_iter)
  for k in K:
    kmeanModel = KMeans(n_clusters=k, n_init=10)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)
  plt.figure(figsize=(16,8))
  plt.plot(K, distortions, 'bx-')
  plt.xlabel('k')
  plt.ylabel('Distortion')
  plt.title('The Elbow Method showing the optimal k')
  plt.show()
```

In [ ]:
```
#Using the elbow method on Cosine distance
elbow_method(df_cosine, 10)
```




In [ ]:
```
#Using the elbow method on Euclidean distance
elbow_method(df_euclidean, 10)
```




First, we have to modify our df in order to keep the first n users and assign our labels to them

Next, we'll use the PCA method in order to reduce the dimensionality of our matrix and plot our clusters

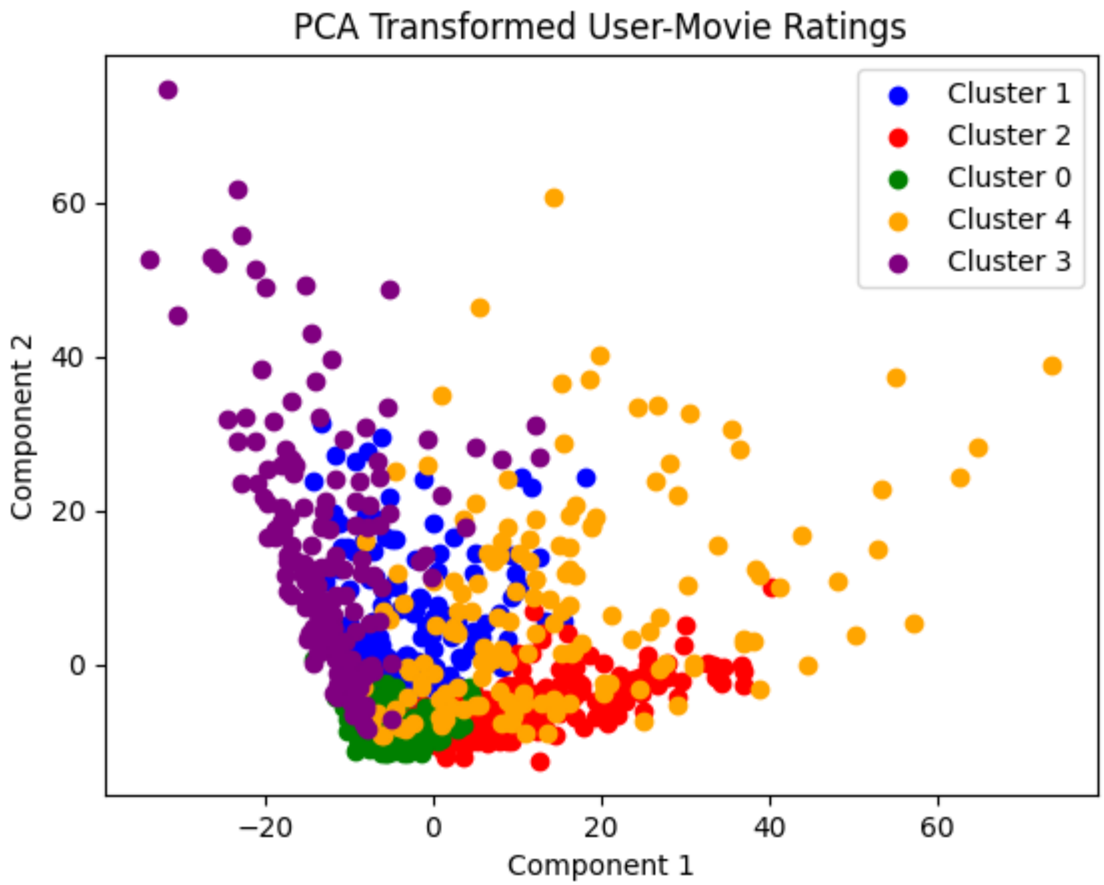
In [ ]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# instantiate StandardScaler and PCA with 2 components for 2D scatter plot
scaler = StandardScaler()
pca = PCA(n_components=2)

# fit and transform the ratings matrix
ratings_pca = pca.fit_transform(ratings_matrix_df)

# print the explained variance ratio for each component
display(pca.explained_variance_ratio_)
```

```
array([0.01590064, 0.01490826])
```

In [ ]:
```
# create a new dataframe with the PCA components and user index
# df_pca = pd.DataFrame(ratings_pca, index=range(0, ratings_matrix_df.shape[0]))
# df_pca['Cluster'] = labels_euclidean
# df_pca
```

In [ ]:
```
#Create a function to transform the DF with PCA to 2 coordinates and create a scatter plot

def plot_pca_cluster(ratings_matrix, n_clusters, label):
    # instantiate StandardScaler and PCA with 2 components for 2D scatter plot
    scaler = StandardScaler()
    pca = PCA(n_components=2)

    # fit and transform the ratings matrix
    ratings_pca = pca.fit_transform(ratings_matrix)

    # # apply K-means clustering
    # kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    # labels = kmeans.fit_predict(ratings_matrix)

    # create a new dataframe with the PCA components and cluster labels
    df_pca = pd.DataFrame(ratings_pca, index=range(0, ratings_matrix.shape[0]), columns=['Component 1', 'Co
    df_pca['Cluster'] = label

    # create a scatter plot of the PCA components with color-coded clusters
    fig, ax = plt.subplots()

    for label, color in zip(df_pca['Cluster'].unique(), ['blue', 'red', 'green', 'orange', 'purple']):
        group = df_pca.groupby('Cluster').get_group(label)
        ax.scatter(group['Component 1'], group['Component 2'], c=color, label=f'Cluster {label}')

    # set the axis labels and title
    ax.set_xlabel('Component 1')
    ax.set_ylabel('Component 2')
    ax.set_title('PCA Transformed User-Movie Ratings')

    # add a legend
    ax.legend()

    # show the plot
    plt.show()
```

In [ ]:
```
#Here we have to set the labels as labels_euclidean or labels_cosine
plot_pca_cluster(ratings_matrix_df, L_CLUSTERS_NUM, labels_euclidean)
```


PCA Transformed User-Movie Ratings
Cluster 1
Cluster 2
Cluster 0
Cluster 4
Cluster 3
Component 2
Component 1
60
40
20
0
−20
0
20
40
60

Να σχολιάσετε την αποτελεσματικότητα των συγκεκριμένων μετρικών στην αποτίμηση της ομοιότητας μεταξύ ενός ζεύγους διανυσμάτων προτιμήσεων χρηστών R_u και R_v.

Για την μετρική της ευκλείδιας απόστασης:

- Η ομοιότητα των χρηστών είναι **αντιστρόφως ανάλογη** της απόστασης μεταξύ τους.
- Για να έχουμε αποτέλεσμα, θα πρέπει να υπάρχει **επικάλυψη μεταξύ των χρηστών.** Πρέπει δηλαδή να έχουν αξιολογήσει κοινές ταινίες.
- Ο υπολογισμός του k-means γίνεται πολύ πιο υπολογιστικά εντατικός λόγω των εκτεταμένων πολλαπλασιασμών πινάκων που εκτελείται.

---

Για την μετρική του συνημιτόνου:

1. Για να έχουμε αποτέλεσμα, θα πρέπει να υπάρχει **επικάλυψη μεταξύ των χρηστών.** Πρέπει δηλαδή να έχουν αξιολογήσει κοινές ταινίες.
2. Ο υπολογισμός του k-means γίνεται πολύ πιο υπολογιστικά εντατικός λόγω των εκτεταμένων πολλαπλασιασμών πινάκων που εκτελείται.
3. Η ομοιότητα των χρηστών μπορεί να υπολογιστεί στην περίπτωση που είναι η γωνία μεταξύ των διανυσμάτων τους από 0 - 90 ως ομοιότητα ενώ από 90 - 180 μπορούμε να εκφράσουμε την αντίθεση των χρηστών. Οπότε σε κάθε περίπτωση η μετρική μας βοηθά να ομαδοποιήσουμε τους χρήστες.

# JACCARD DISTANCE

Η απόσταση Jaccard απομετρά τη **διαφορετικότητα** μεταξύ δύο συνόλων (στην περίπτωσή μας δύο χρηστών).

- Στην περίπτωση που η τομή των δύο χρηστών γίνει μηδέν (δεν υπάρχουν δηλαδή κοινά αξιολογήσιμες ταινίες) η διαφορετικότητα των χρηστών παίρνει τη μέγιστη τιμή της, 1
- Η διαφορετικότητα των χρηστών θα γίνει **ελάχιστη** όταν η *τομή* των δύο χρηστών είναι ίση με την *ένωσή* τους, όταν δηλαδή τα δύο σύνολα γίνουν *ίσα*
- Μπορεί να χρησιμοποιηθεί για τη σύγκριση της ομοιότητας οποιουδήποτε είδους δεδομένων, συμπεριλαμβανομένων δεδομένων χρονοσειρών, φωτογραφιών, κειμένου και εικόνων.

---

Κάποια από τα μειονεκτήματα της ανωτέρω μετρικής είναι τα ακόλουθα:

---

- **Απουσία "βαρών"**: Η απόσταση Jaccard εξετάζει μόνο την παρουσία ή την απουσία αξιολογήσεων για κάθε χρήστη και δεν λαμβάνει υπόψη τις πραγματικές τιμές αξιολόγησης. Μπορεί δηλαδή η *διαφορετικότητα*, η τιμή δηλαδή που θα προκύψει από την απόσταση Jaccard δύο χρηστών να είναι ελάχιστη, εάν έχουν αξιολογήσει τις ίδιες ταινίες ακόμα και αν ο ένας τις έχει αξιολογήσει με 5 και ο άλλος με 1.
- **Αραιότητα αξιολογήσεων**: Για παράδειγμα, εάν δύο χρήστες έχουν αξιολογήσει μόνο έναν μικρό αριθμό ταινιών, είναι πιθανό να μην έχουν αξιολογήσει καμία από τις ίδιες ταινίες, άρα η τομή τους θα είναι μηδέν, με αποτέλεσμα η *διαφορετικότητά* τους να είναι μέγιστη, ακόμη και αν οι προτιμήσεις τους για τις ταινίες είναι στην πραγματικότητα αρκετά παρόμοιες.

In [ ]:
```
jaccard_dist = 1 - W
jaccard_df = pd.DataFrame(jaccard_dist)

def kmeans_clustering(jaccard_dist, L):
    # Initialize k-means object
    kmeans = KMeans(n_clusters=L)

    # Fit the k-means object to the Jaccard distance matrix
    kmeans.fit(jaccard_dist)

    return kmeans.labels_

def create_jaccard_labels_cached(jaccard_dist, L: int):
    npy_file = os.path.join(DATAFOLDER_PATH, "L_K_DEPEND_jaccard_labels.npy")
    if os.path.exists(npy_file):
        jaccard_labels: np_typing.NDArray = np.load(npy_file, allow_pickle=True)
        return jaccard_labels
    else:
        jaccard_labels = kmeans_clustering(jaccard_dist, L)
        np.save(npy_file, jaccard_labels, allow_pickle=True, fix_imports=True)
        return jaccard_labels


jaccard_labels = create_jaccard_labels_cached(jaccard_dist, L_CLUSTERS_NUM)
display('jaccard_df', jaccard_df)
display('jaccard_dist', jaccard_dist)
display('jaccard_labels', jaccard_labels)
```

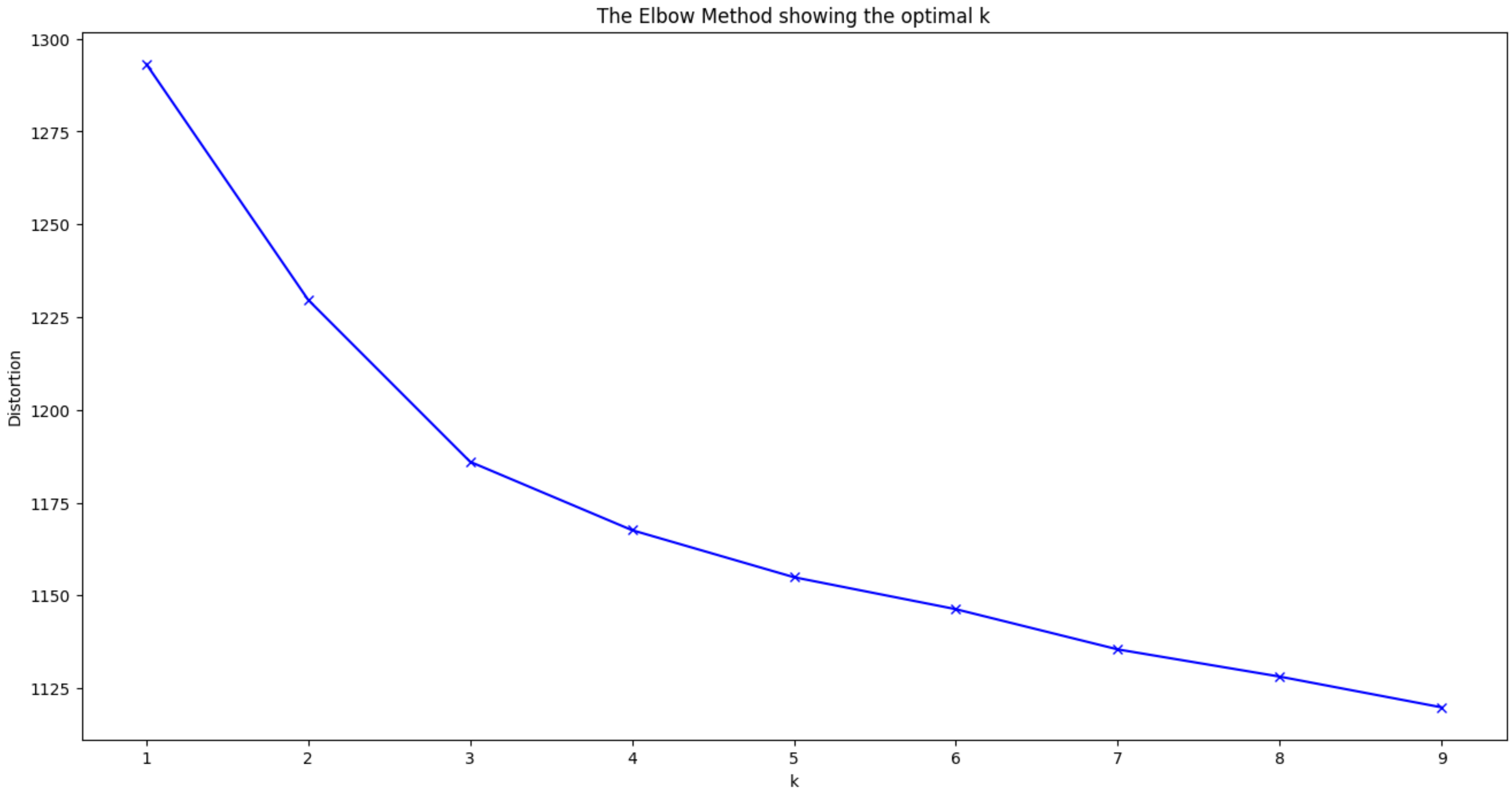
```
/home/thanos/.pyenv/versions/3.11.2/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explic
itly to suppress the warning
  warnings.warn(
'jaccard_df'
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1069 | 1070 | 1071 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.000000 | 0.983721 | 0.997717 | 0.991031 | 0.990971 | 1.000000 | 0.991266 | 1.000000 | 0.997738 | 0.989455 | ... | 0.977186 | 1.0 | 0.994444 |
| **1** | 0.983721 | 0.000000 | 0.993421 | 0.993651 | 0.983819 | 0.990826 | 0.978261 | 0.990741 | 0.996764 | 0.995455 | ... | 0.984962 | 1.0 | 0.995098 |
| **2** | 0.997717 | 0.993421 | 0.000000 | 0.964286 | 0.983923 | 0.978462 | 0.993921 | 0.968652 | 0.993548 | 0.993197 | ... | 0.977387 | 1.0 | 0.977667 |
| **3** | 0.991031 | 0.993651 | 0.964286 | 0.000000 | 0.984472 | 0.988201 | 0.988166 | 0.985075 | 0.996894 | 0.995585 | ... | 0.995192 | 1.0 | 0.995249 |
| **4** | 0.990971 | 0.983819 | 0.983923 | 0.984472 | 0.000000 | 0.994083 | 0.981982 | 0.978788 | 0.980892 | 0.984270 | ... | 0.967662 | 1.0 | 0.985507 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1074** | 1.000000 | 0.996815 | 0.993651 | 0.996942 | 1.000000 | 0.997059 | 0.997050 | 1.000000 | 1.000000 | 0.997788 | ... | 0.995169 | 1.0 | 0.968137 |
| **1075** | 0.995614 | 0.984375 | 0.984472 | 0.984985 | 0.987915 | 0.991379 | 0.994253 | 0.991304 | 0.996970 | 0.989083 | ... | 0.975962 | 1.0 | 0.985882 |
| **1076** | 0.997717 | 0.993421 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.996785 | 0.995475 | ... | 0.990074 | 1.0 | 1.000000 |
| **1077** | 0.998267 | 1.000000 | 0.986395 | 0.997812 | 0.995585 | 0.997872 | 1.000000 | 1.000000 | 1.000000 | 0.991349 | ... | 0.994475 | 1.0 | 0.994526 |
| **1078** | 0.990020 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.992021 | 0.998039 | ... | 0.978448 | 1.0 | 0.993697 |

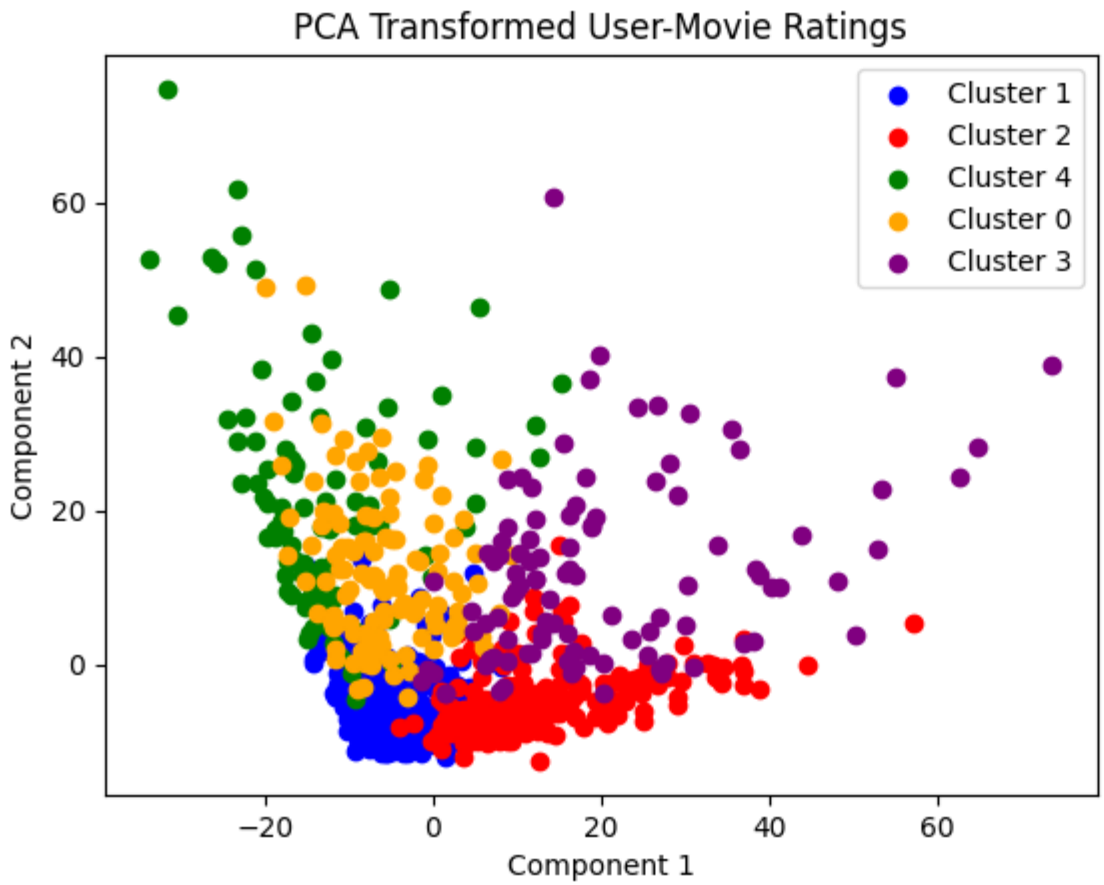
1079 rows × 1079 columns

```
'jaccard_dist'
array([[0.        , 0.98372093, 0.99771689, ..., 0.99771689, 0.9982669 ,
        0.99001996],
       [0.98372093, 0.        , 0.99342105, ..., 0.99342105, 1.        ,
        1.        ],
       [0.99771689, 0.99342105, 0.        , ..., 1.        , 0.98639456,
        1.        ],
       ...,
       [0.99771689, 0.99342105, 1.        , ..., 0.        , 0.99775785,
        0.9973262 ],
       [0.9982669 , 1.        , 0.98639456, ..., 0.99775785, 0.        ,
        1.        ],
       [0.99001996, 1.        , 1.        , ..., 0.9973262 , 1.        ,
        0.        ]])
'jaccard_labels'
array([1, 1, 2, ..., 1, 1, 1], dtype=int32)
```

In [ ]:
```
elbow_method(jaccard_df, 10)
```




In [ ]:
```
plot_pca_cluster(ratings_matrix_df, L_CLUSTERS_NUM, jaccard_labels)
```




# Neural Network

## Pre - processing

We will first start by seperating our Users according to the Cluster they've been assigned to, using the Jaccard distance on the K-Means algorithm.

We do this by creating a df containing the ratings of each user and the Cluster it belongs to.

In [ ]:
```
ratings_matrix_clustered = ratings_matrix_df

ratings_matrix_clustered['Cluster'] = jaccard_labels

ratings_matrix_clustered
```

Out[ ]:

| Movie | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 68075 | 68076 | 68077 | 68078 | 68079 | 68080 | 68081 | 68082 | 68083 | Cluster |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **User** | | | | | | | | | | | | | | | | | | | | | |
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1074** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| **1075** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3 |
| **1076** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1 |
| **1077** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 |
| **1078** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 |

1079 rows × 68085 columns

In [ ]:
```
#We sort the Labels of the Clusters from 0 to 4
# clusters = sorted(ratings_matrix_clustered.Cluster.unique())
# clusters = ratings_matrix_clustered.Cluster.unique()

# #We save each Cluster in an array where each position is for the same Cluster
# clustered_DFs: list[np_typing.NDArray] = []
# for cluster in clusters:
#     groupby_result = ratings_matrix_clustered.groupby('Cluster').get_group(cluster)
#     clustered_DFs.append(groupby_result.to_numpy())
# display('clustered_DFs', clustered_DFs)



# For each cluster find the ratings and the jaccard distances
# Each cluster_ratings array has shape (#cluster users, #total movies)
# Each cluster_jaccard_distances array has shape (#cluster users, #total users)
# Each cluster_users_indexes array has a set with len(#cluster users)

def calculate_clusters_ratings_jaccard_distances_users_indexes(L_CLUSTERS_NUM: int, jaccard_labels: np_typ:
    clusters_ratings_dict = { i: [] for i in range(L_CLUSTERS_NUM)}
    clusters_jaccard_distances_dict = { i: [] for i in range(L_CLUSTERS_NUM)}
    clusters_users_indexes_dict: dict[int, set[int]] = { i: set() for i in range(L_CLUSTERS_NUM)}

    for i in range(jaccard_labels.shape[0]):
        label = jaccard_labels[i]
        cluster_ratings = clusters_ratings_dict[label]
        cluster_jaccard_distances = clusters_jaccard_distances_dict[label]
        cluster_users_indexes = clusters_users_indexes_dict[label]

        cluster_ratings.append(ratings_matrix_array[i])
        cluster_jaccard_distances.append(jaccard_dist[i])
        cluster_users_indexes.add(i)


    clusters_ratings_list: list[np_typing.NDArray] = []
    for key in clusters_ratings_dict:
        cluster_ratings = np.array(clusters_ratings_dict[key])
        clusters_ratings_list.append(cluster_ratings)


    clusters_jaccard_distances_list: list[np_typing.NDArray] = []
    for key in clusters_jaccard_distances_dict:
```

```
    for key in clusters_jaccard_distances_dict:
        cluster_jaccard_distances = np.array(clusters_jaccard_distances_dict[key])
        clusters_jaccard_distances_list.append(cluster_jaccard_distances)

    clusters_users_indexes_list: list[set[int]] = []
    for key in clusters_users_indexes_dict:
        clusters_users_indexes_list.append(clusters_users_indexes_dict[key])


    # clusters_ratings cotains the cluster_ratings array for each cluster
    clusters_ratings = np.array(clusters_ratings_list)
    # clusters_jaccard_distances cotains the cluster_jaccard_distances array for each cluster
    cluster_jaccard_distances = np.array(clusters_jaccard_distances_list)
    # clusters_users_indexes cotains the users indexes that belong in this cluster
    clusters_users_indexes = np.array(clusters_users_indexes_list)
    return clusters_ratings, cluster_jaccard_distances, clusters_users_indexes


clusters_ratings, clusters_jaccard_distances, clusters_users_indexes = calculate_clusters_ratings_jaccard_d
display(f'clusters_ratings[0].shape: {clusters_ratings[0].shape}')
display(f'clusters_jaccard_distances[0].shape: {clusters_jaccard_distances[0].shape}')
display(f'len(clusters_users_indexes[0]): {len(clusters_users_indexes[0])}')
display('clusters_ratings', clusters_ratings)
display('clusters_jaccard_distances', clusters_jaccard_distances)
display('clusters_users_indexes', clusters_users_indexes)
```

```
/tmp/ipykernel_3203/920973544.py:52: VisibleDeprecationWarning: Creating an ndarray from ragged nested seq
uences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is depre
cated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  clusters_ratings = np.array(clusters_ratings_list)
/tmp/ipykernel_3203/920973544.py:54: VisibleDeprecationWarning: Creating an ndarray from ragged nested seq
uences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is depre
cated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  cluster_jaccard_distances = np.array(clusters_jaccard_distances_list)
'clusters_ratings[0].shape: (118, 68084)'
'clusters_jaccard_distances[0].shape: (118, 1079)'
'len(clusters_users_indexes[0]): 118'
'clusters_ratings'
```

```
array([array([[0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              ...,
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.]]),
       array([[0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              ...,
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.]]),
       array([[0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              ...,
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.]]),
       array([[0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              ...,
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.]]),
       array([[0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              ...,
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.],
              [0., 0., 0., ..., 0., 0., 0.]])], dtype=object)
'clusters_jaccard_distances'
```

```
array([array([[0.99266055, 0.99029126, 0.99519231, ..., 1.        , 0.9963964 ,
                0.99585921],
               [0.98091603, 0.98992443, 0.99501247, ..., 0.99501247, 0.99815157,
                0.98047722],
               [0.9869403 , 0.98263027, 0.99512195, ..., 0.99512195, 0.99635701,
                0.99580713],
               ...,
               [0.99546485, 1.        , 0.98039216, ..., 0.99029126, 0.97732426,
                0.9973545 ],
               [0.98630137, 0.98148148, 0.98425197, ..., 0.9921875 , 0.99426386,
                0.99111111],
               [1.        , 0.99681529, 0.99365079, ..., 0.99044586, 0.99337748,
                1.        ]])                                                     ,
       array([[0.        , 0.98372093, 0.99771689, ..., 0.99771689, 0.9982669 ,
                0.99001996],
               [0.98372093, 0.        , 0.99342105, ..., 0.99342105, 1.        ,
                1.        ],
               [0.99103139, 0.99365079, 0.96428571, ..., 1.        , 0.99781182,
                1.        ],
               ...,
               [0.99771689, 0.99342105, 1.        , ..., 0.        , 0.99775785,
                0.9973262 ],
               [0.9982669 , 1.        , 0.98639456, ..., 0.99775785, 0.        ,
                1.        ],
               [0.99001996, 1.        , 1.        , ..., 0.9973262 , 1.        ,
                0.        ]])                                                     ,
       array([[0.99771689, 0.99342105, 0.        , ..., 1.        , 0.98639456,
                1.        ],
               [0.99097065, 0.98381877, 0.98392283, ..., 1.        , 0.99558499,
                1.        ],
               [0.99126638, 0.97826087, 0.99392097, ..., 1.        , 1.        ,
                1.        ],
               ...,
               [0.98695652, 0.9939577 , 0.96604938, ..., 0.99399399, 0.99576271,
                0.99750623],
               [0.98350515, 0.99159664, 0.99722992, ..., 0.99722992, 0.998     ,
                0.99295775],
               [0.99444444, 0.99509804, 0.97766749, ..., 1.        , 0.99452555,
                0.99369748]])                                                     ,
       array([[0.99125874, 0.97695853, 1.        , ..., 0.99095023, 0.99311532,
                0.99608611],
               [0.96806387, 0.976     , 0.9921671 , ..., 0.9895288 , 0.99232246,
```

```
                 0.99778761],
                [0.98499062, 0.99259259, 1.        , ..., 0.98765432, 0.99634369,
                 0.99789916],
                ...,
                [0.97718631, 0.98496241, 0.97738693, ..., 0.99007444, 0.99447514,
                 0.97844828],
                [0.99342105, 0.97484277, 0.99076923, ..., 0.98765432, 1.        ,
                 0.99746193],
                [0.99561404, 0.984375  , 0.98447205, ..., 0.99384615, 0.97356828,
                 0.99489796]])                                                   ,
         array([[1.        , 0.98480243, 1.        , ..., 0.99401198, 0.9978903 ,
                 0.98997494],
                [0.99622642, 0.98214286, 1.        , ..., 0.97704082, 0.99441341,
                 0.99354839],
                [0.99438202, 0.98746867, 0.99004975, ..., 0.985     , 0.98698885,
                 0.99575372],
                ...,
                [0.98850575, 0.99234694, 0.9872449 , ..., 0.98465473, 0.98484848,
                 0.98468271],
                [0.98830409, 0.97883598, 0.99741602, ..., 0.99481865, 0.99619048,
                 0.99113082],
                [0.99558499, 0.97452229, 0.9875    , ..., 0.99378882, 0.99347826,
                 0.99226804]])                                                   ],
        dtype=object)
 'clusters_users_indexes'
```

```
array([{1030, 522, 1034, 528, 1050, 540, 547, 1061, 559, 1074, 564, 565, 56, 571, 577, 583, 587, 595, 598,
602, 91, 94, 612, 619, 631, 122, 637, 645, 648, 649, 651, 659, 661, 665, 667, 157, 680, 685, 686, 694, 18
3, 701, 702, 708, 198, 710, 711, 712, 714, 729, 731, 220, 734, 224, 225, 741, 743, 745, 757, 761, 251, 76
3, 766, 768, 258, 776, 782, 786, 787, 799, 802, 294, 810, 301, 813, 305, 818, 310, 822, 823, 318, 837, 83
8, 850, 853, 856, 859, 351, 864, 868, 359, 361, 364, 881, 884, 376, 894, 898, 902, 404, 918, 415, 416, 93
0, 931, 422, 939, 951, 967, 978, 471, 991, 481, 994, 996, 1000, 1002, 1014},
       {0, 1, 3, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 21, 24, 25, 26, 27, 28, 31, 34, 35, 36, 3
7, 39, 41, 43, 44, 45, 47, 51, 54, 55, 58, 59, 60, 61, 62, 63, 65, 66, 68, 69, 72, 75, 78, 79, 81, 83, 84,
87, 88, 89, 90, 95, 97, 98, 102, 104, 106, 107, 108, 110, 112, 113, 118, 120, 123, 124, 125, 126, 127, 12
8, 129, 130, 132, 134, 137, 138, 141, 143, 144, 147, 148, 149, 151, 155, 159, 161, 163, 164, 166, 167, 16
8, 169, 173, 176, 180, 181, 182, 185, 188, 189, 192, 195, 197, 204, 206, 208, 209, 210, 211, 214, 216, 21
7, 219, 221, 226, 227, 228, 230, 231, 233, 235, 238, 240, 241, 242, 243, 244, 246, 247, 248, 249, 252, 25
3, 255, 262, 264, 265, 268, 269, 270, 271, 272, 273, 274, 275, 279, 281, 283, 286, 287, 288, 289, 290, 29
1, 292, 293, 296, 297, 300, 302, 306, 308, 309, 312, 313, 314, 315, 317, 319, 320, 321, 322, 325, 326, 32
7, 329, 333, 334, 335, 336, 337, 339, 340, 341, 344, 347, 348, 352, 353, 357, 360, 362, 366, 367, 368, 37
1, 372, 373, 374, 377, 378, 379, 380, 381, 385, 386, 387, 388, 389, 391, 395, 396, 398, 399, 400, 405, 40
6, 407, 411, 412, 417, 420, 421, 424, 425, 426, 429, 430, 431, 435, 437, 438, 441, 442, 444, 447, 448, 44
9, 451, 452, 454, 455, 458, 459, 460, 461, 462, 463, 464, 467, 468, 469, 470, 473, 475, 476, 477, 479, 48
0, 483, 484, 487, 488, 493, 494, 495, 496, 497, 499, 501, 502, 504, 506, 507, 509, 511, 512, 514, 515, 51
8, 523, 525, 529, 530, 534, 535, 537, 538, 539, 542, 544, 548, 550, 552, 554, 555, 558, 560, 561, 562, 56
3, 575, 576, 578, 579, 584, 586, 589, 590, 592, 593, 594, 597, 599, 600, 603, 604, 605, 606, 607, 608, 60
9, 610, 613, 614, 616, 617, 618, 620, 622, 623, 624, 625, 628, 632, 633, 635, 636, 638, 639, 640, 641, 64
3, 644, 650, 653, 654, 655, 660, 663, 664, 672, 673, 675, 676, 681, 682, 687, 690, 691, 692, 693, 697, 69
8, 699, 700, 703, 704, 707, 713, 717, 718, 720, 722, 724, 728, 730, 732, 733, 735, 736, 737, 740, 744, 74
7, 748, 750, 751, 754, 756, 758, 759, 762, 764, 765, 767, 769, 771, 773, 774, 775, 777, 778, 779, 780, 78
1, 785, 788, 792, 793, 795, 797, 800, 801, 805, 806, 807, 811, 814, 817, 819, 820, 821, 824, 825, 826, 83
2, 834, 835, 839, 841, 843, 844, 845, 849, 851, 852, 855, 858, 860, 863, 865, 866, 867, 869, 871, 872, 87
3, 874, 875, 877, 878, 879, 880, 882, 883, 886, 888, 889, 891, 893, 895, 896, 900, 904, 905, 906, 909, 91
5, 916, 919, 921, 922, 923, 924, 925, 926, 927, 928, 929, 934, 935, 936, 938, 940, 943, 944, 946, 948, 94
9, 952, 953, 955, 956, 957, 958, 959, 960, 963, 965, 966, 969, 970, 975, 976, 979, 980, 981, 984, 986, 98
7, 988, 995, 997, 998, 999, 1001, 1003, 1004, 1005, 1006, 1008, 1015, 1019, 1020, 1021, 1022, 1023, 1025,
1026, 1032, 1035, 1036, 1037, 1040, 1042, 1043, 1044, 1047, 1049, 1051, 1053, 1054, 1057, 1058, 1059, 107
0, 1072, 1076, 1077, 1078},
       {2, 4, 1028, 6, 519, 520, 521, 1031, 527, 16, 531, 20, 22, 23, 536, 1046, 29, 30, 1055, 32, 33, 54
5, 546, 38, 1062, 40, 42, 46, 1071, 48, 49, 50, 52, 567, 568, 57, 569, 574, 64, 67, 580, 581, 70, 71, 73,
74, 76, 77, 591, 80, 82, 596, 85, 86, 601, 92, 93, 96, 99, 100, 101, 103, 105, 109, 111, 114, 115, 116, 11
7, 629, 119, 121, 634, 131, 133, 646, 135, 136, 139, 140, 652, 145, 146, 662, 152, 153, 154, 156, 671, 16
0, 162, 165, 170, 174, 177, 178, 179, 186, 187, 190, 191, 193, 194, 196, 199, 200, 201, 202, 203, 715, 20
5, 716, 207, 212, 213, 725, 215, 218, 222, 223, 229, 232, 234, 236, 237, 239, 752, 245, 760, 250, 254, 25
6, 257, 259, 260, 263, 266, 267, 276, 277, 790, 282, 284, 299, 311, 316, 830, 323, 324, 328, 842, 331, 33
2, 847, 848, 338, 342, 854, 345, 350, 354, 876, 369, 375, 382, 384, 390, 392, 393, 394, 397, 911, 401, 40
3, 408, 409, 410, 413, 419, 423, 428, 942, 432, 433, 434, 945, 436, 439, 445, 446, 450, 964, 453, 457, 46
```

```
5, 466, 985, 474, 482, 485, 490, 491, 492, 498, 500, 503, 505, 508, 510},
       {1027, 516, 517, 1033, 524, 1039, 532, 1045, 1052, 541, 543, 1056, 551, 1063, 553, 1064, 1065, 556,
557, 1067, 1068, 1069, 1073, 1075, 566, 570, 572, 573, 582, 585, 588, 611, 615, 621, 142, 656, 658, 150, 6
66, 669, 670, 677, 678, 679, 171, 172, 684, 175, 689, 695, 705, 706, 726, 742, 749, 261, 789, 278, 791, 28
5, 798, 295, 808, 809, 298, 812, 303, 304, 815, 307, 828, 831, 836, 330, 857, 862, 355, 358, 365, 890, 38
3, 897, 907, 913, 402, 917, 418, 427, 950, 443, 961, 974, 983, 472, 478, 990, 486, 1009, 1016, 1017},
       {1024, 513, 1029, 526, 1038, 1041, 533, 1048, 1060, 549, 1066, 53, 626, 627, 630, 642, 647, 657, 66
8, 158, 674, 683, 688, 184, 696, 709, 719, 721, 723, 727, 738, 739, 746, 753, 755, 770, 772, 783, 784, 28
0, 794, 796, 803, 804, 816, 827, 829, 833, 840, 846, 343, 346, 349, 861, 356, 870, 363, 370, 885, 887, 89
2, 899, 901, 903, 908, 910, 912, 914, 920, 414, 932, 933, 937, 941, 947, 440, 954, 962, 456, 968, 971, 97
2, 973, 977, 982, 989, 992, 993, 489, 1007, 1010, 1011, 1012, 1013, 1018}],
      dtype=object)
```

In [ ]:
```
from sklearn.neighbors import NearestNeighbors

def find_nearest_neighbors_using_jaccard(K_NEIGHBORS_NUM: int, cluster_jaccard_distances: np_typing.NDArray
    nearest_neighbors_list: list[np_typing.NDArray] = []
    for row_index in range(cluster_jaccard_distances.shape[0]):
        cluster_jaccard_row = cluster_jaccard_distances[row_index]

        ## set the distance of the users that are the same a index to a value higher thatn 1
        for j in range(cluster_jaccard_row.shape[0]):
            if j == row_index or not(j in cluster_users_indexes):
                cluster_jaccard_row[j] = 2

        ## find the k smallest indexes
        k_nearest_indexes = np.argpartition(cluster_jaccard_row, K_NEIGHBORS_NUM)
        # k_nearest_indexes = k_nearest_indexes[k_nearest_indexes != row_index]
        nearest_neighbors_list.append(k_nearest_indexes[:K_NEIGHBORS_NUM])

    return np.array(nearest_neighbors_list)


def create_clusters_nearest_neighbors_cached(K_NEIGHBORS_NUM: int, clusters_jaccard_distances: np_typing.ND
    #Set the npy file that will store the clusters_nearest_neighbors
    npy_file = os.path.join(DATAFOLDER_PATH, "L_K_DEPEND_clusters_nearest_neighbors.npy")
    if os.path.exists(npy_file):
        clusters_nearest_neighbors: np_typing.NDArray[np.float64] = np.load(npy_file, allow_pickle=True)
        return clusters_nearest_neighbors
    else:
        clusters_nearest_neighbors_list: list[np_typing.NDArray] = []
        for index in range(clusters_jaccard_distances.shape[0]):
            cluster_jaccard_distances = clusters_jaccard_distances[index]
            # cluster_ratings_binary = np.where(cluster_ratings > 0, 1, 0)
            # nearest_neihbors = find_nearest_neighbors(cluster_ratings_binary, k)
            nearest_neihbors = find_nearest_neighbors_using_jaccard(K_NEIGHBORS_NUM, clusters_jaccard_dista
            display('nearest_neihbors', nearest_neihbors)
            clusters_nearest_neighbors_list.append(nearest_neihbors)


        clusters_nearest_neighbors = np.array(clusters_nearest_neighbors_list)
        np.save(npy_file, clusters_nearest_neighbors, allow_pickle=True, fix_imports=True)
        return clusters_nearest_neighbors
```

```
clusters_nearest_neighbors = create_clusters_nearest_neighbors_cached(K_NEIGHBORS_NUM, clusters_jaccard_dis
display('clusters_nearest_neighbors', clusters_nearest_neighbors)

# # instantiate the NearestNeighbors model with the custom distance metric
# model = NearestNeighbors(n_neighbors=k, algorithm='brute', metric=custom_distance)

# cluster_ratings = clustered_DFs[1]
# # fit the model on the ratings matrix
# model.fit(cluster_ratings)

# # find the k-nearest neighbors for each user
# k_nearest_neighbors = {}
# for i in range(cluster_ratings.shape[0]):
#     _, indices = model.kneighbors([cluster_ratings[i]], n_neighbors=k+1) # get indices of k+1 most simila
#     # if we want to get the distance for each pair of users
#     # neighbors = [(index, custom_distance(ratings[i], ratings[index])) for index in indices[0] if index
#     neighbors = [index for index in indices[0] if index != i]
#  # exclude the user itself
#     k_nearest_neighbors[i] = neighbors

# # We save our k_nearest_neighbors as a dict where for each user, we get the
# # most similar of their users. This will allow us to
# # Create a NN where the INPUT: will be the ratings of similar users
# # and OUTPUT: the rating of the user we currently have.
```

```
'nearest_neihbors'
```

```
array([[ 667,   56,  743,  838,  224,  659],
       [ 631,   91,  743,  761,  376,  310],
       [ 637,  743,   94,  710,  667,  818],
       [ 122,  645,  631,  571,  667,  577],
       [ 157,  868,  416,  881,  631,  665],
       [ 183, 1050,  351,  528,  734,  708],
       [ 667,  198,  710,  743,  637,  838],
       [ 881,  220,  708,  598,  522,  894],
       [ 810,  224,  659,  667,  318,  838],
       [ 710,  225,  838,  694,  810,  743],
       [ 251,  645,  667,  731,  649,  659],
       [ 258,  602,  364,  930, 1050,  522],
       [ 294,  565,  649,  602,  364,  351],
       [ 659,  310,  301,  763,  602,  631],
       [ 743,  710,  305,  708,  881,  637],
       [ 631,  310,  743,  619,  659,  761],
       [ 694,  659,  224,  318,  667,  602],
       [ 361, 1050,  708,  351,  565,  571],
       [ 359,  587, 1050,  602,  680,  850],
       [ 361,  710,  818,  743,  637,  837],
       [ 602,  649,  364,  595,  294,  631],
       [ 631,  310,  376,   91,  761,  818],
       [ 587,  364,  823,  571,  404,  602],
       [ 694,  850,  881,  415,  837,  931],
       [ 710,  637,  702,  416,  743,  818],
       [ 743,  422,  837,  763,  881,  761],
       [ 648,  471,  667,  710,  571,  565],
       [ 481,  837,  422,  881,  743,  710],
       [ 648,  743,  708,  361,  522,  818],
       [ 528,  183,  587,  881,  818,  648],
       [ 702,  540,  838,  741,  743,  710],
       [ 667,  547,  659,  838,  810,  645],
       [ 686,  565,  649,  559,  708,  648],
       [ 631,  310,  602,  564,  547,   94],
       [ 565,  649,  645,  667,  294,  637],
       [ 631,  602,  351,  587,  708,  571],
       [ 694,  818,  637,  743,  577,  710],
       [ 659,  602,  583,  731,  422,  631],
       [ 648,  587,  571,  818,  708,  602],
       [ 645,  595,  667,  602,  631,  659],
       [ 708,  598,  220,  881,  837,  818],
       [ 649,  602,  667,  645,  659,  364],
```

```
       [ 612,  881,  768,  710,  818,  708],
       [ 743,  850,  761,  310,  619,  931],
       [ 631,  667,  818,  310,  602,  376],
       [ 637,  743,  710,  702,  818,  994],
       [ 648,  649,  602,  645,  667,  659],
       [ 667,  645,  648,  708,  649,  710],
       [ 649,  602,  648,  645,  667,  565],
       [ 702,  651,  743,  710,  637,  818],
       [ 667,  645,  659,  602,  838,  224],
       [ 708,  667,  661,  602,  645,  659],
       [ 665,  743,  823,  881,  782,  868],
       [ 667,  645,  838,  702,  602,  659],
       [ 710,  602,  680,  667,  198,  631],
       [ 708,  685,  743,  310,  667,  637],
       [ 602,  649,  686,  559,  587,  708],
       [ 710,  694,  994,  667,  838,  743],
       [ 310,  376,  761,  701,  631,  856],
       [ 637,  710,  743,  702,  540,  838],
       [ 708,  743,  818,  881,  710,  637],
       [ 710,  743,  637,  702,  361,  838],
       [ 711,  838,  702,  416,  823,  741],
       [ 838,  712,  710,  667,  818,  361],
       [ 714,  838,  667,  702,  648,  665],
       [ 787,  729,  649,  637,  565,  710],
       [ 637,  710,  731,  743,  667,  659],
       [ 734,  818, 1050,  708,  598,  823],
       [ 710,  741,  702,  743,  540,  838],
       [ 743,  710,  637,  702,  361,  994],
       [ 710,  818,  361,  743,  745,  637],
       [ 310,  757,  761,  931,  850,  768],
       [ 619,  422,  743,  761,  837,  631],
       [ 422,  881,  743,  837,  763,  361],
       [ 850,  881,  708,  766,  902,  818],
       [ 881,  743,  768,  787,  818,  708],
       [ 776,  645,  667,  602,  595,  648],
       [ 637,  881,  837,  743,  782,  710],
       [ 786,  837,  881,  884,  787,  743],
       [ 787,  743,  637,  710,  881,  768],
       [ 799,  648,  602,  649,  694,  837],
       [ 802,  743,  648,  667,  645,  881],
       [ 710,  667,  838,  810,  818,  224],
       [ 813,  818,  837,  763,  310,  931],
```

```
        [ 710,  818,  743,  637,  708,  810],
        [ 822,  743,  710,  708,  838,  361],
        [ 665,  710,  823,  708,  305,  602],
        [ 881,  743,  422,  837,  361,  763],
        [ 838,  702,  710,  667,  743,  540],
        [ 881,  837,  850,  931,  743,  894],
        [ 853,  743,  361,  710,  422,  837],
        [ 710,  994,  743,  856,  637,  702],
        [ 859,  708,  598,  881,  743,  884],
        [ 931,  894,  540,  864,  743,  702],
        [ 838,  702,  868,  416,  637,  665],
        [ 881,  422,  708,  837,  743,  884],
        [ 881,  884,  708,  743,  837,  637],
        [ 894,  818,  881,  837,  708,  743],
        [ 710,  838,  702,  994,  898,  743],
        [ 522,  902,  708,  881,  649,  571],
        [ 918,  850,  818,  931,  930,  881],
        [ 881,  930,  708, 1050,  648,  918],
        [ 837,  850,  881,  931,  422,  761],
        [ 694,  939,  577,  710,  602,  665],
        [ 577,  951,  850,  598,  881,  743],
        [ 967,  761,  837,  931,  416,  708],
        [ 708,  931,  939,  978,  799,  881],
        [ 743,  991,  710,  637,  745,  708],
        [ 743,  994,  710,  637,  702,  818],
        [ 996, 1074,  881,  708,  305, 1000],
        [1000,  708,  667, 1074,  602,  996],
        [ 631,   94,  305, 1034, 1002,  310],
        [ 786, 1050,  708, 1014,  649,  565],
        [1030,  799,  602, 1074,  996,  631],
        [1034,  782,  631,  310,  305,  694],
        [1050,  351,  708,  884,  734, 1074],
        [ 305,  850,  310,  602, 1061,  918],
        [ 996,  708, 1074, 1050,  351,  645]])
'nearest_neihbors'
array([[ 480,  412,  592,  976,  214,  733],
        [ 412,   75,  107, 1025,  817,  463],
        [ 424,  618,   31,    3,   61,  764],
        ...,
        [ 663,  534, 1076,  955,  374,  997],
        [ 306,  980, 1077,  451,  826,  929],
        [ 420, 1078,  579,  214,  673,  374]])
```

```
'nearest_neihbors'
array([[  32,  190,    2,  115,   93,   33],
       [ 218,  259,    4,  156,  239,  354],
       [   6,  331,  382,  254,  116,  205],
       ...,
       [ 716, 1055,  236,  218,  945,  194],
       [ 531,  725, 1062,  662,  199,  260],
       [1071,  945,  260,  154,  257,  725]])
'nearest_neihbors'
```

```
array([[ 142, 1056,  726,  656,  150,  285],
       [ 656,  150,  726, 1056,  588,  142],
       [ 171,  726,  658,  890,  917,  656],
       [ 285,  261,  172,  298,  142,  517],
       [1063,  175,  656,  809, 1068, 1065],
       [ 285,  298,  261,  172,  365,  831],
       [ 478,  990,  427,  278,  621,  298],
       [1056,  669,  917,  836,  285,  990],
       [ 556,  295,  472,  582,  402,  543],
       [ 298,  478,  573,  142,  443,  261],
       [ 556,  303,  365,  705,  669,  472],
       [ 142,  836,  285,  304,  808,  656],
       [ 307, 1068, 1063, 1065, 1067,  689],
       [ 573,  478,  809,  330,  689,  418],
       [ 556,  418,  355,  749,  678,  809],
       [ 358, 1068, 1063, 1067, 1069, 1065],
       [ 556,  261,  365,  669,  142,  285],
       [ 556,  383,  573,  543,  582,  295],
       [ 556,  472,  402,  582,  295,  543],
       [ 585, 1069,  478,  418, 1063, 1067],
       [ 427,  478,  726,  588,  278,  573],
       [ 285,  556,  573,  478,  443,  298],
       [ 556,  472,  543,  669,  295,  705],
       [ 478,  551,  418,  573,  443,  330],
       [ 486,  726,  142,  656, 1056,  573],
       [ 836, 1039,  656,  516,  990,  532],
       [ 142,  656,  172,  517,  295,  285],
       [ 695,  689,  142,  524,  551,  656],
       [ 532,  295,  656,  472,  791,  150],
       [ 541,  669,  588,  472,  615,  677],
       [ 669,  295,  543,  582,  556,  472],
       [ 478, 1063, 1067,  551, 1068, 1069],
       [ 142,  556,  553,  582,  726,  295],
       [ 365,  556,  543,  582,  472,  557],
       [ 557,  556,  543,  295,  472,  666],
       [ 815,  689,  566, 1056,  726,  917],
       [ 570, 1065, 1069, 1063, 1067, 1068],
       [ 556,  809,  749,  572,  472,  615],
       [ 573,  478,  809,  330,  621,  383],
       [ 582,  472,  556,  543,  295,  402],
       [1069,  585, 1068, 1063, 1067,  418],
       [ 588,  669,  689, 1056,  472,  684],
```

```
[ 836, 1068,  611, 1063,  828,  831],
[ 472,  615,  582,  669,  556,  543],
[ 621,  726,  836,  809,  917,  990],
[ 656,  726,  150, 1056,  588,  809],
[ 658, 1056,  726,  171,  582, 1017],
[ 669,  666, 1056,  472,  588,  566],
[ 472,  669,  588,  689,  684,  677],
[ 749, 1067, 1063, 1069,  670, 1068],
[1065, 1068,  689,  677, 1063,  669],
[ 588, 1067,  418,  678, 1065, 1063],
[ 418, 1069,  551,  679,  585,  669],
[ 684,  669,  556,  588,  472,  543],
[ 677,  689, 1063, 1065, 1068, 1067],
[ 261,  789,  524,  695,  656,  836],
[ 705,  472,  669,  677,  303,  670],
[1075,  726,  706,  656,  836, 1056],
[1056, 1075,  815,  726,  656,  890],
[1069, 1068,  742, 1067,  917,  836],
[1063, 1068, 1065,  749, 1067, 1069],
[ 726,  990,  789,  669, 1069,  897],
[1065, 1069, 1067, 1063,  791, 1068],
[ 585,  689,  798,  418, 1068,  791],
[ 808,  689,  304,  669,  836, 1056],
[ 809,  621, 1067,  656, 1065,  478],
[1063,  812, 1067, 1068, 1065, 1069],
[ 917, 1056,  815,  566,  726, 1075],
[1067, 1063,  828, 1068, 1065, 1069],
[ 611,  726,  831,  836,  261, 1039],
[1039, 1065, 1068,  836, 1067, 1069],
[1065,  726,  857, 1039, 1068,  836],
[1065,  862, 1039, 1068, 1067, 1063],
[ 890,  726,  656,  917,  990, 1056],
[ 726, 1064,  897, 1067, 1068, 1039],
[ 689,  726,  907,  974, 1075, 1056],
[1056,  913,  726,  890, 1033,  307],
[1056,  917,  726,  836,  566,  285],
[1045,  950,  330, 1068, 1069, 1033],
[1068, 1056,  961, 1027, 1063, 1065],
[ 974, 1056,  726, 1027,  836,  656],
[1039,  261,  983, 1068,  669, 1067],
[ 990,  726, 1056,  836,  621,  285],
[1068, 1009, 1056,  836, 1039,  669],
```

```
       [1016,  621,  418,  611, 1067,  285],
       [1056, 1033,  726, 1017,  689, 1075],
       [1063, 1039, 1027, 1065, 1068,  689],
       [ 726, 1033, 1056,  897, 1017,  307],
       [1039, 1063, 1065, 1067, 1068,  836],
       [1063, 1069, 1065, 1045, 1068,  418],
       [1052, 1063, 1067, 1065, 1069, 1068],
       [ 566, 1056,  726,  917,  815,  142],
       [1065, 1063, 1067, 1068, 1069,  812],
       [1068, 1064, 1063, 1067, 1065, 1069],
       [1067, 1069, 1065, 1063, 1068,  812],
       [1065, 1069, 1063, 1067, 1068,  812],
       [1067, 1065, 1063, 1068, 1069,  812],
       [1069, 1063, 1065, 1067, 1068,  570],
       [1073, 1065, 1067, 1069, 1063, 1039],
       [1075,  706, 1056,  726,  897,  917]])
'nearest_neihbors'
```

```
array([[ 158,  941,   53,  770,  630, 1024],
       [ 549,  630,  158,  941,  973,  954],
       [ 184, 1007,  630,  739,  954,  549],
       [ 549, 1007,  280,  829,  954,  739],
       [ 549,  954,  343,  784, 1007,  892],
       [ 647,  630,  346,  982,  709,  954],
       [1007,  349,  993,  784,  973,  549],
       [ 954,  356,  973,  784,  158,  549],
       [ 941,  982,  363,  657,  630, 1007],
       [1007,  630,  954,  370,  739,  941],
       [ 954,  414,  973,  772, 1007,  549],
       [ 440, 1007,  549,  941, 1018,  954],
       [ 721,  456,  549,  784,  158,  954],
       [ 489,  158, 1007,  549,  630,  954],
       [1007,  954, 1024,  343,  982,  513],
       [ 356,  526,  982,  549,  954,  630],
       [ 971, 1018, 1024,  627,  533,  738],
       [ 343,  630, 1007,  784,  954,  549],
       [ 626,  549,  158,  954, 1007,  892],
       [ 627,  630, 1024,  971,  992,  549],
       [ 158,  954,  630,  549,  941,  971],
       [ 723,  642, 1038,  696,  647,  738],
       [ 709,  803, 1038, 1007,  647,  954],
       [ 657,  630, 1007,  982, 1018,  941],
       [1018,  941,  668,  158, 1024,  753],
       [ 739,  954,  674,  647, 1007,  892],
       [ 647,  954,  630,  683,  738,  971],
       [1018,  688, 1007, 1024,  941,  647],
       [ 696,  642,  723, 1038,  971,  738],
       [ 709,  738,  630, 1007,  941,  647],
       [ 719,  549,  739, 1010,  954,  647],
       [ 721,  982,  932, 1007,  941,  158],
       [ 642,  723,  971,  696,  738, 1038],
       [ 971,  954,  727,  630,  932, 1024],
       [1024,  753,  738, 1007,  971, 1018],
       [ 739, 1007,  549,  755,  630,  954],
       [ 746,  892,  972,  941, 1007,  982],
       [ 753,  738, 1018, 1024, 1007,  954],
       [ 158, 1007,  755,  630,  549,  954],
       [ 941,  770,  932,  833,  973,  158],
       [ 772,  549,  356,  954,  158,  343],
       [ 657,  549,  783,  932,  158,  941],
```

```
[ 549,  954,  343,  784,  892, 1007],
[ 414,  772,  794,  739,  356,  158],
[1024,  796,  816,  738, 1018,  971],
[ 803,  968, 1007,  971,  647,  738],
[ 804,  992,  709,  158,  972,  941],
[ 971, 1024, 1018,  630,  816,  941],
[ 827, 1007, 1024, 1018,  630,  941],
[ 829,  993, 1007,  887,  549,  954],
[ 833,  158,  973,  356,  549,  941],
[1007, 1024, 1018,  840,  657,  738],
[ 549,  846,  971, 1024,  738,  755],
[ 861,  158,  973,  356,  941,  549],
[1007,  630,  954,  870,  549,  982],
[ 954,  630,  885,  971,  549,  738],
[ 549,  954, 1007,  887,  784, 1018],
[ 549,  343,  892,  784,  954, 1007],
[ 829,  899,  770,  993,  933,  908],
[ 901,  349,  932,  783,  914,  982],
[ 755,  739,  903, 1024,  738,  630],
[ 908, 1024,  941,  982, 1007,  549],
[1007,  657,  910,  954,  630,  887],
[ 908,  738, 1007,  912,  829, 1018],
[ 914,  784,  549,  932,  356,  954],
[ 920,  738,  829,  954,  908,  941],
[ 932,  941,  968,  783,  158,  657],
[ 755,  158,  933,  630, 1007,  899],
[ 647,  755,  739, 1007,  954,  937],
[ 941,  630, 1018,  158, 1024,  982],
[ 630,  947,  972, 1007,  833,  549],
[ 954,  549, 1007,  784,  630,  343],
[1024,  630,  962, 1007,  738,  971],
[ 968,  549,  982,  803,  932,  954],
[1018,  971,  630,  816,  738,  954],
[ 972,  630, 1007,  657,  982,  954],
[ 973,  158,  356,  941,  784,  549],
[1007,  977,  739,  755,  158,  630],
[ 630,  941,  657,  982, 1007,  356],
[1038,  989,  982,  343,  892,  414],
[ 627, 1018,  992,  755, 1007,  630],
[ 993,  829, 1007,  349,  784,  941],
[1018,  630,  954, 1024, 1007,  549],
[1048, 1007, 1010,  982,  972,  657],
```

```
       [1018,  630,  738, 1011, 1024,  971],
       [1012,  982,  630,  887,  954,  356],
       [ 954, 1013,  630,  549,  158, 1007],
       [1024, 1018,  971, 1007,  738,  941],
       [ 738,  753, 1007, 1018, 1024,  941],
       [ 158, 1029,  549,  343, 1007,  489],
       [1038,  642,  647, 1007,  709,  696],
       [1041,  657,  941, 1024, 1007,  709],
       [1024, 1048,  549, 1010, 1007,  982],
       [1007,  657, 1060,  941,  630, 1024],
       [1066,  829, 1007,  993,  954,  549]])
```

```
/tmp/ipykernel_3203/811023617.py:38: VisibleDeprecationWarning: Creating an ndarray from ragged nested seq
uences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is depre
cated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  clusters_nearest_neighbors = np.array(clusters_nearest_neighbors_list)
'clusters_nearest_neighbors'
```

```
array([array([[ 667,   56,  743,  838,  224,  659],
              [ 631,   91,  743,  761,  376,  310],
              [ 637,  743,   94,  710,  667,  818],
              [ 122,  645,  631,  571,  667,  577],
              [ 157,  868,  416,  881,  631,  665],
              [ 183, 1050,  351,  528,  734,  708],
              [ 667,  198,  710,  743,  637,  838],
              [ 881,  220,  708,  598,  522,  894],
              [ 810,  224,  659,  667,  318,  838],
              [ 710,  225,  838,  694,  810,  743],
              [ 251,  645,  667,  731,  649,  659],
              [ 258,  602,  364,  930, 1050,  522],
              [ 294,  565,  649,  602,  364,  351],
              [ 659,  310,  301,  763,  602,  631],
              [ 743,  710,  305,  708,  881,  637],
              [ 631,  310,  743,  619,  659,  761],
              [ 694,  659,  224,  318,  667,  602],
              [ 361, 1050,  708,  351,  565,  571],
              [ 359,  587, 1050,  602,  680,  850],
              [ 361,  710,  818,  743,  637,  837],
              [ 602,  649,  364,  595,  294,  631],
              [ 631,  310,  376,   91,  761,  818],
              [ 587,  364,  823,  571,  404,  602],
              [ 694,  850,  881,  415,  837,  931],
              [ 710,  637,  702,  416,  743,  818],
              [ 743,  422,  837,  763,  881,  761],
              [ 648,  471,  667,  710,  571,  565],
              [ 481,  837,  422,  881,  743,  710],
              [ 648,  743,  708,  361,  522,  818],
              [ 528,  183,  587,  881,  818,  648],
              [ 702,  540,  838,  741,  743,  710],
              [ 667,  547,  659,  838,  810,  645],
              [ 686,  565,  649,  559,  708,  648],
              [ 631,  310,  602,  564,  547,   94],
              [ 565,  649,  645,  667,  294,  637],
              [ 631,  602,  351,  587,  708,  571],
              [ 694,  818,  637,  743,  577,  710],
              [ 659,  602,  583,  731,  422,  631],
              [ 648,  587,  571,  818,  708,  602],
              [ 645,  595,  667,  602,  631,  659],
              [ 708,  598,  220,  881,  837,  818],
              [ 649,  602,  667,  645,  659,  364],
```

```
[ 612,  881,  768,  710,  818,  708],
[ 743,  850,  761,  310,  619,  931],
[ 631,  667,  818,  310,  602,  376],
[ 637,  743,  710,  702,  818,  994],
[ 648,  649,  602,  645,  667,  659],
[ 667,  645,  648,  708,  649,  710],
[ 649,  602,  648,  645,  667,  565],
[ 702,  651,  743,  710,  637,  818],
[ 667,  645,  659,  602,  838,  224],
[ 708,  667,  661,  602,  645,  659],
[ 665,  743,  823,  881,  782,  868],
[ 667,  645,  838,  702,  602,  659],
[ 710,  602,  680,  667,  198,  631],
[ 708,  685,  743,  310,  667,  637],
[ 602,  649,  686,  559,  587,  708],
[ 710,  694,  994,  667,  838,  743],
[ 310,  376,  761,  701,  631,  856],
[ 637,  710,  743,  702,  540,  838],
[ 708,  743,  818,  881,  710,  637],
[ 710,  743,  637,  702,  361,  838],
[ 711,  838,  702,  416,  823,  741],
[ 838,  712,  710,  667,  818,  361],
[ 714,  838,  667,  702,  648,  665],
[ 787,  729,  649,  637,  565,  710],
[ 637,  710,  731,  743,  667,  659],
[ 734,  818, 1050,  708,  598,  823],
[ 710,  741,  702,  743,  540,  838],
[ 743,  710,  637,  702,  361,  994],
[ 710,  818,  361,  743,  745,  637],
[ 310,  757,  761,  931,  850,  768],
[ 619,  422,  743,  761,  837,  631],
[ 422,  881,  743,  837,  763,  361],
[ 850,  881,  708,  766,  902,  818],
[ 881,  743,  768,  787,  818,  708],
[ 776,  645,  667,  602,  595,  648],
[ 637,  881,  837,  743,  782,  710],
[ 786,  837,  881,  884,  787,  743],
[ 787,  743,  637,  710,  881,  768],
[ 799,  648,  602,  649,  694,  837],
[ 802,  743,  648,  667,  645,  881],
[ 710,  667,  838,  810,  818,  224],
[ 813,  818,  837,  763,  310,  931],
```

```
        [ 710,  818,  743,  637,  708,  810],
        [ 822,  743,  710,  708,  838,  361],
        [ 665,  710,  823,  708,  305,  602],
        [ 881,  743,  422,  837,  361,  763],
        [ 838,  702,  710,  667,  743,  540],
        [ 881,  837,  850,  931,  743,  894],
        [ 853,  743,  361,  710,  422,  837],
        [ 710,  994,  743,  856,  637,  702],
        [ 859,  708,  598,  881,  743,  884],
        [ 931,  894,  540,  864,  743,  702],
        [ 838,  702,  868,  416,  637,  665],
        [ 881,  422,  708,  837,  743,  884],
        [ 881,  884,  708,  743,  837,  637],
        [ 894,  818,  881,  837,  708,  743],
        [ 710,  838,  702,  994,  898,  743],
        [ 522,  902,  708,  881,  649,  571],
        [ 918,  850,  818,  931,  930,  881],
        [ 881,  930,  708, 1050,  648,  918],
        [ 837,  850,  881,  931,  422,  761],
        [ 694,  939,  577,  710,  602,  665],
        [ 577,  951,  850,  598,  881,  743],
        [ 967,  761,  837,  931,  416,  708],
        [ 708,  931,  939,  978,  799,  881],
        [ 743,  991,  710,  637,  745,  708],
        [ 743,  994,  710,  637,  702,  818],
        [ 996, 1074,  881,  708,  305, 1000],
        [1000,  708,  667, 1074,  602,  996],
        [ 631,   94,  305, 1034, 1002,  310],
        [ 786, 1050,  708, 1014,  649,  565],
        [1030,  799,  602, 1074,  996,  631],
        [1034,  782,  631,  310,  305,  694],
        [1050,  351,  708,  884,  734, 1074],
        [ 305,  850,  310,  602, 1061,  918],
        [ 996,  708, 1074, 1050,  351,  645]]),
 array([[ 480,  412,  592,  976,  214,  733],
        [ 412,   75,  107, 1025,  817,  463],
        [ 424,  618,   31,    3,   61,  764],
        ...,
        [ 663,  534, 1076,  955,  374,  997],
        [ 306,  980, 1077,  451,  826,  929],
        [ 420, 1078,  579,  214,  673,  374]]),
 array([[  32,  190,    2,  115,   93,   33],
```

```
       [ 218,  259,    4,  156,  239,  354],
       [   6,  331,  382,  254,  116,  205],
       ...,
       [ 716, 1055,  236,  218,  945,  194],
       [ 531,  725, 1062,  662,  199,  260],
       [1071,  945,  260,  154,  257,  725]]),
array([[ 142, 1056,  726,  656,  150,  285],
       [ 656,  150,  726, 1056,  588,  142],
       [ 171,  726,  658,  890,  917,  656],
       [ 285,  261,  172,  298,  142,  517],
       [1063,  175,  656,  809, 1068, 1065],
       [ 285,  298,  261,  172,  365,  831],
       [ 478,  990,  427,  278,  621,  298],
       [1056,  669,  917,  836,  285,  990],
       [ 556,  295,  472,  582,  402,  543],
       [ 298,  478,  573,  142,  443,  261],
       [ 556,  303,  365,  705,  669,  472],
       [ 142,  836,  285,  304,  808,  656],
       [ 307, 1068, 1063, 1065, 1067,  689],
       [ 573,  478,  809,  330,  689,  418],
       [ 556,  418,  355,  749,  678,  809],
       [ 358, 1068, 1063, 1067, 1069, 1065],
       [ 556,  261,  365,  669,  142,  285],
       [ 556,  383,  573,  543,  582,  295],
       [ 556,  472,  402,  582,  295,  543],
       [ 585, 1069,  478,  418, 1063, 1067],
       [ 427,  478,  726,  588,  278,  573],
       [ 285,  556,  573,  478,  443,  298],
       [ 556,  472,  543,  669,  295,  705],
       [ 478,  551,  418,  573,  443,  330],
       [ 486,  726,  142,  656, 1056,  573],
       [ 836, 1039,  656,  516,  990,  532],
       [ 142,  656,  172,  517,  295,  285],
       [ 695,  689,  142,  524,  551,  656],
       [ 532,  295,  656,  472,  791,  150],
       [ 541,  669,  588,  472,  615,  677],
       [ 669,  295,  543,  582,  556,  472],
       [ 478, 1063, 1067,  551, 1068, 1069],
       [ 142,  556,  553,  582,  726,  295],
       [ 365,  556,  543,  582,  472,  557],
       [ 557,  556,  543,  295,  472,  666],
       [ 815,  689,  566, 1056,  726,  917],
```

```
[ 570, 1065, 1069, 1063, 1067, 1068],
[ 556,  809,  749,  572,  472,  615],
[ 573,  478,  809,  330,  621,  383],
[ 582,  472,  556,  543,  295,  402],
[1069,  585, 1068, 1063, 1067,  418],
[ 588,  669,  689, 1056,  472,  684],
[ 836, 1068,  611, 1063,  828,  831],
[ 472,  615,  582,  669,  556,  543],
[ 621,  726,  836,  809,  917,  990],
[ 656,  726,  150, 1056,  588,  809],
[ 658, 1056,  726,  171,  582, 1017],
[ 669,  666, 1056,  472,  588,  566],
[ 472,  669,  588,  689,  684,  677],
[ 749, 1067, 1063, 1069,  670, 1068],
[1065, 1068,  689,  677, 1063,  669],
[ 588, 1067,  418,  678, 1065, 1063],
[ 418, 1069,  551,  679,  585,  669],
[ 684,  669,  556,  588,  472,  543],
[ 677,  689, 1063, 1065, 1068, 1067],
[ 261,  789,  524,  695,  656,  836],
[ 705,  472,  669,  677,  303,  670],
[1075,  726,  706,  656,  836, 1056],
[1056, 1075,  815,  726,  656,  890],
[1069, 1068,  742, 1067,  917,  836],
[1063, 1068, 1065,  749, 1067, 1069],
[ 726,  990,  789,  669, 1069,  897],
[1065, 1069, 1067, 1063,  791, 1068],
[ 585,  689,  798,  418, 1068,  791],
[ 808,  689,  304,  669,  836, 1056],
[ 809,  621, 1067,  656, 1065,  478],
[1063,  812, 1067, 1068, 1065, 1069],
[ 917, 1056,  815,  566,  726, 1075],
[1067, 1063,  828, 1068, 1065, 1069],
[ 611,  726,  831,  836,  261, 1039],
[1039, 1065, 1068,  836, 1067, 1069],
[1065,  726,  857, 1039, 1068,  836],
[1065,  862, 1039, 1068, 1067, 1063],
[ 890,  726,  656,  917,  990, 1056],
[ 726, 1064,  897, 1067, 1068, 1039],
[ 689,  726,  907,  974, 1075, 1056],
[1056,  913,  726,  890, 1033,  307],
[1056,  917,  726,  836,  566,  285],
```

```
        [1045,  950,  330, 1068, 1069, 1033],
        [1068, 1056,  961, 1027, 1063, 1065],
        [ 974, 1056,  726, 1027,  836,  656],
        [1039,  261,  983, 1068,  669, 1067],
        [ 990,  726, 1056,  836,  621,  285],
        [1068, 1009, 1056,  836, 1039,  669],
        [1016,  621,  418,  611, 1067,  285],
        [1056, 1033,  726, 1017,  689, 1075],
        [1063, 1039, 1027, 1065, 1068,  689],
        [ 726, 1033, 1056,  897, 1017,  307],
        [1039, 1063, 1065, 1067, 1068,  836],
        [1063, 1069, 1065, 1045, 1068,  418],
        [1052, 1063, 1067, 1065, 1069, 1068],
        [ 566, 1056,  726,  917,  815,  142],
        [1065, 1063, 1067, 1068, 1069,  812],
        [1068, 1064, 1063, 1067, 1065, 1069],
        [1067, 1069, 1065, 1063, 1068,  812],
        [1065, 1069, 1063, 1067, 1068,  812],
        [1067, 1065, 1063, 1068, 1069,  812],
        [1069, 1063, 1065, 1067, 1068,  570],
        [1073, 1065, 1067, 1069, 1063, 1039],
        [1075,  706, 1056,  726,  897,  917]]),
array([[ 158,  941,   53,  770,  630, 1024],
        [ 549,  630,  158,  941,  973,  954],
        [ 184, 1007,  630,  739,  954,  549],
        [ 549, 1007,  280,  829,  954,  739],
        [ 549,  954,  343,  784, 1007,  892],
        [ 647,  630,  346,  982,  709,  954],
        [1007,  349,  993,  784,  973,  549],
        [ 954,  356,  973,  784,  158,  549],
        [ 941,  982,  363,  657,  630, 1007],
        [1007,  630,  954,  370,  739,  941],
        [ 954,  414,  973,  772, 1007,  549],
        [ 440, 1007,  549,  941, 1018,  954],
        [ 721,  456,  549,  784,  158,  954],
        [ 489,  158, 1007,  549,  630,  954],
        [1007,  954, 1024,  343,  982,  513],
        [ 356,  526,  982,  549,  954,  630],
        [ 971, 1018, 1024,  627,  533,  738],
        [ 343,  630, 1007,  784,  954,  549],
        [ 626,  549,  158,  954, 1007,  892],
        [ 627,  630, 1024,  971,  992,  549],
```

```
[ 158,  954,  630,  549,  941,  971],
[ 723,  642, 1038,  696,  647,  738],
[ 709,  803, 1038, 1007,  647,  954],
[ 657,  630, 1007,  982, 1018,  941],
[1018,  941,  668,  158, 1024,  753],
[ 739,  954,  674,  647, 1007,  892],
[ 647,  954,  630,  683,  738,  971],
[1018,  688, 1007, 1024,  941,  647],
[ 696,  642,  723, 1038,  971,  738],
[ 709,  738,  630, 1007,  941,  647],
[ 719,  549,  739, 1010,  954,  647],
[ 721,  982,  932, 1007,  941,  158],
[ 642,  723,  971,  696,  738, 1038],
[ 971,  954,  727,  630,  932, 1024],
[1024,  753,  738, 1007,  971, 1018],
[ 739, 1007,  549,  755,  630,  954],
[ 746,  892,  972,  941, 1007,  982],
[ 753,  738, 1018, 1024, 1007,  954],
[ 158, 1007,  755,  630,  549,  954],
[ 941,  770,  932,  833,  973,  158],
[ 772,  549,  356,  954,  158,  343],
[ 657,  549,  783,  932,  158,  941],
[ 549,  954,  343,  784,  892, 1007],
[ 414,  772,  794,  739,  356,  158],
[1024,  796,  816,  738, 1018,  971],
[ 803,  968, 1007,  971,  647,  738],
[ 804,  992,  709,  158,  972,  941],
[ 971, 1024, 1018,  630,  816,  941],
[ 827, 1007, 1024, 1018,  630,  941],
[ 829,  993, 1007,  887,  549,  954],
[ 833,  158,  973,  356,  549,  941],
[1007, 1024, 1018,  840,  657,  738],
[ 549,  846,  971, 1024,  738,  755],
[ 861,  158,  973,  356,  941,  549],
[1007,  630,  954,  870,  549,  982],
[ 954,  630,  885,  971,  549,  738],
[ 549,  954, 1007,  887,  784, 1018],
[ 549,  343,  892,  784,  954, 1007],
[ 829,  899,  770,  993,  933,  908],
[ 901,  349,  932,  783,  914,  982],
[ 755,  739,  903, 1024,  738,  630],
[ 908, 1024,  941,  982, 1007,  549],
```

```
       [1007,  657,  910,  954,  630,  887],
       [ 908,  738, 1007,  912,  829, 1018],
       [ 914,  784,  549,  932,  356,  954],
       [ 920,  738,  829,  954,  908,  941],
       [ 932,  941,  968,  783,  158,  657],
       [ 755,  158,  933,  630, 1007,  899],
       [ 647,  755,  739, 1007,  954,  937],
       [ 941,  630, 1018,  158, 1024,  982],
       [ 630,  947,  972, 1007,  833,  549],
       [ 954,  549, 1007,  784,  630,  343],
       [1024,  630,  962, 1007,  738,  971],
       [ 968,  549,  982,  803,  932,  954],
       [1018,  971,  630,  816,  738,  954],
       [ 972,  630, 1007,  657,  982,  954],
       [ 973,  158,  356,  941,  784,  549],
       [1007,  977,  739,  755,  158,  630],
       [ 630,  941,  657,  982, 1007,  356],
       [1038,  989,  982,  343,  892,  414],
       [ 627, 1018,  992,  755, 1007,  630],
       [ 993,  829, 1007,  349,  784,  941],
       [1018,  630,  954, 1024, 1007,  549],
       [1048, 1007, 1010,  982,  972,  657],
       [1018,  630,  738, 1011, 1024,  971],
       [1012,  982,  630,  887,  954,  356],
       [ 954, 1013,  630,  549,  158, 1007],
       [1024, 1018,  971, 1007,  738,  941],
       [ 738,  753, 1007, 1018, 1024,  941],
       [ 158, 1029,  549,  343, 1007,  489],
       [1038,  642,  647, 1007,  709,  696],
       [1041,  657,  941, 1024, 1007,  709],
       [1024, 1048,  549, 1010, 1007,  982],
       [1007,  657, 1060,  941,  630, 1024],
       [1066,  829, 1007,  993,  954,  549]])], dtype=object)
```

# Creating the NN

# 2. NEURAL NETWORK

In [ ]:
```
def create_nn_original_df(ratings_matrix_array: np_typing.NDArray, cluster_ratings: np_typing.NDArray, nea
    # create user_ratings_list
    user_index_list: list[int] = []
    movie_index_list: list[int] = []
    user_ratings_list: list[int] = []
    for i in range(nearest_neighbors.shape[0]):
        user_index_list.extend([i for ratings in cluster_ratings[i]])
        movie_index_list.extend([j for j in range(cluster_ratings[i].shape[0])])
        user_ratings_list.extend(cluster_ratings[i])

    # create neighbors list
    neighbors_list: list[list[int]] = []
    for i in range(nearest_neighbors.shape[1]):
        neighbor_ratings_list: list[int] = []

        for j in range(nearest_neighbors.shape[0]):
            neighbor = nearest_neighbors[j][i]
            # nearest neighbors have indexs for the ratings_matrix_array up to total users
            # so use ratings_matrix_array instead of cluster_ratings.
            # We have previously ensured that all neighbors belong to this clusters
            neighbor_ratings_list.extend(ratings_matrix_array[neighbor])

        neighbors_list.append(neighbor_ratings_list)


    nn_origin_df = pd.DataFrame()
    nn_origin_df['USER_INDEX'] = user_index_list
    nn_origin_df['MOVIE_INDEX'] = movie_index_list
    nn_origin_df['USER_RATINGS'] = user_ratings_list

    for i in range(len(neighbors_list)):
        neighbor_ratings_list: list[int] = neighbors_list[i]
        nn_origin_df[NEIGHBOURS_COLUMNS[i]] = neighbor_ratings_list


    return nn_origin_df


def create_clusters_nn_original_dfs_cached(ratings_matrix_array: np_typing.NDArray, clusters_ratings: np_ty
    for cluster_index in range(len(clusters_ratings)):
        pickle_file = os.path.join(DATAFOLDER_PATH, f"L_K_DEPEND_clusters_nn_original_dfs_{cluster_index}.p
        if os.path.exists(pickle_file):
```

```
        if os.path.exists(pickle_file):
            print('exists')
        else:
            cluster_ratings = clusters_ratings[cluster_index]
            nearest_neighbors = clusters_nearest_neighbors[cluster_index]

            nn_origin_df = create_nn_original_df(ratings_matrix_array, cluster_ratings, nearest_neighbors,
            nn_origin_df.to_pickle(pickle_file)
            nn_origin_df = None


ratings_normalize_factor: float = ratings_matrix_df.max().max()
# NOTE: ratings_normalize_factor == 1 (not normalizing data) produces better results from normalizing data
ratings_normalize_factor = 1

NEIGHBOURS_COLUMNS = [f'NEIGHBOR_RATINGS_{i}' for i in range(K_NEIGHBORS_NUM)]

create_clusters_nn_original_dfs_cached(ratings_matrix_array, clusters_ratings, clusters_nearest_neighbors,
```

In [ ]:

```
# Define functions to create and train a linear regression model

def create_nn_filtered_normalized_df(nn_origin_df: pd.DataFrame, NEIGHBOURS_COLUMNS: list[str], ratings_no
    # filter the rows that have USER_RATINGS == 0
    nn_filtered_df = nn_origin_df.copy()[nn_origin_df['USER_RATINGS'] != 0]

    # filter the rows that have all NEIGHBOR_RATINGS == 0
    filter_neighbors_query: str = f'{NEIGHBOURS_COLUMNS[0]} != 0'
    for neihbor_column in NEIGHBOURS_COLUMNS[1:]:
        filter_neighbors_query += f' | {neihbor_column} != 0'

    nn_filtered_df = nn_filtered_df.query(filter_neighbors_query)

    # create filtered normalized df
    nn_filtered_normalized_df = nn_filtered_df.copy()
    columns_to_normalize = ['USER_RATINGS']
    columns_to_normalize.extend(NEIGHBOURS_COLUMNS)
    nn_filtered_normalized_df[columns_to_normalize] = nn_filtered_normalized_df[columns_to_normalize] / rat
    display('Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating')
    display(nn_filtered_normalized_df)
    display(nn_filtered_normalized_df.describe())
    return nn_filtered_df, nn_filtered_normalized_df
```

```
def create_model(my_learning_rate: float, input_shape: tuple):
    """Create and compile a simple linear regression model."""
    # Most simple tf.keras models are sequential.
    model = tf.keras.models.Sequential()


    # Add the layer containing the feature columns to the model.
    model.add(tf.keras.layers.InputLayer(input_shape=input_shape))

    model.add(tf.keras.layers.Masking(
        mask_value=0
    ))

    # Implement L2 regularization in the first hidden layer.
    model.add(tf.keras.layers.Dense(units=12,
                                    activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l2(0.01),
                                    name='Hidden1'))

    # Implement L2 regularization in the second hidden layer.
    model.add(tf.keras.layers.Dense(units=24,
                                    activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l2(0.01),
                                    name='Hidden2'))

    # Implement L2 regularization in the third hidden layer.
    model.add(tf.keras.layers.Dense(units=48,
                                    activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l2(0.01),
                                    name='Hidden3'))

    # Define the output layer.
    model.add(tf.keras.layers.Dense(units=1,
                                    name='Output'))

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=my_learning_rate),
                  loss="mean_squared_error",
                  metrics=[tf.keras.metrics.MeanSquaredError(), tf.keras.metrics.MeanAbsoluteError()])
    return model


def train_model(model: tf.keras.models.Sequential, X_train: np_typing.NDArray, Y_train: np_typing.NDArray,
```

```
    """Train the model by feeding it X_train."""

    # Features as a numpy array
    history = model.fit(x=X_train, y=Y_train, batch_size=batch_size, epochs=epochs, shuffle=True)

    # The list of epochs is stored separately from the rest of history.
    epochs = history.epoch

    # To track the progression of training, gather a snapshot
    # of the model's mean squared error at each epoch.
    hist = pd.DataFrame(history.history)
    mse = hist["mean_squared_error"]
    mae = hist["mean_absolute_error"]

    return epochs, mse, mae


def evaluate_model(model: tf.keras.models.Sequential, X_test: np_typing.NDArray, Y_test: np_typing.NDArray,
    """Evaluate the model against the X_test"""

    # Features as a numpy array
    return model.evaluate(x = X_test, y = Y_test, batch_size=batch_size)


def predict_model(model: tf.keras.models.Sequential, X_origin: np_typing.NDArray, batch_size: int=1):
    """Predict the model with the X_origin"""

    # Features as a numpy array
    return model.predict(x = X_origin, batch_size=batch_size)


def plot_the_loss_curve(epochs, mse_or_mae, is_mse: bool, filename: str):
    """Plot a curve of loss vs. epoch."""

    plt.figure()
    plt.xlabel("Epoch")
    ylabel = 'Train Mean Squared Error' if is_mse else 'Train Mean Absolute Error'
    plt.ylabel(ylabel)

    plt.plot(epochs, mse_or_mae, label="Loss")
    plt.legend()
    plt.ylim([mse_or_mae.min()*0.95, mse_or_mae.max() * 0.6])
    plt.savefig(os.path.join(RESULTS_PATH, filename))
```

```
    plt.savefig(os.path.join(RESULTS_PATH, filename))
    plt.show()



def calculate_real_mse_mae(origin_ratings: np_typing.NDArray, predictions: np_typing.NDArray):
    """Calculate the real mse and mae comparing real ratings and predictions."""

    n = 0
    absolute_sum = 0
    squared_sum = 0
    for i in range(origin_ratings.shape[0]):
        # take into consideration only non 0 ratings
        if origin_ratings[i] != 0.0:
            n += 1
            abs_value = abs(origin_ratings[i] - predictions[i])
            absolute_sum += abs_value
            squared_sum += math.sqrt(abs_value)


    mse = absolute_sum / n
    mae = squared_sum / n
    return mse, mae


def create_train_evaluate_neural_network(ratings_normalize_factor, nn_origin_df: pd.DataFrame, nn_filtered_
    train_df, test_df = train_test_split(nn_filtered_normalized_df, test_size=0.2, random_state=42)
    train_df = pd.DataFrame(train_df)
    test_df = pd.DataFrame(test_df)

    display('train_df', train_df)
    display('test_df', test_df)

    # The following variables are the hyperparameters.
    learning_rate = 0.001
    epochs = 64
    batch_size = 128

    # define the feature columns
    # Establish the model's topography.
    X_train = train_df[NEIGHBOURS_COLUMNS].to_numpy()
    Y_train = train_df['USER_RATINGS'].to_numpy()
    input_shape = (X_train.shape[1],)
```

```
    my_model = create_model(learning_rate, input_shape)

    # Train the model on the normalized training set.
    display('Training the model with the train_df')
    epochs, train_mse_series, train_mae_series = train_model(my_model, X_train, Y_train, epochs, batch_size
    train_mse = train_mse_series.iloc[-1]
    train_mae = train_mae_series.iloc[-1]
    plot_the_loss_curve(epochs, train_mse_series, True, f'cluster{cluster_index}_mse.png')
    plot_the_loss_curve(epochs, train_mae_series, False, f'cluster{cluster_index}_mae.png')

    display('Evaluating the model against the test_df')
    X_test = test_df[NEIGHBOURS_COLUMNS].to_numpy()
    Y_test = test_df['USER_RATINGS'].to_numpy()
    test_loss, test_mse, test_mae = evaluate_model(my_model, X_test, Y_test, batch_size)

    display('Predicting the nn_origin_df and comparing with the initial data')
    X_origin = nn_origin_df[NEIGHBOURS_COLUMNS].to_numpy()
    predictions_normalized = predict_model(my_model, X_origin, batch_size)
    # turn list of sinle item lists to a sigle list with floats
    predictions_normalized = np.array([prediction_normalized[0] for prediction_normalized in predictions_no
    predictions = predictions_normalized * ratings_normalize_factor
    display('predictions', predictions)
    display('predictions.shape', predictions.shape)

    real_ratings = nn_origin_df['USER_RATINGS'].to_numpy()
    display('real_ratings', real_ratings)

    # Save csv results
    real_ratings_predictions_df = nn_origin_df.copy()
    real_ratings_predictions_df['USER_RATINGS'] = real_ratings
    real_ratings_predictions_df.insert(3, 'PREDICTIONS', predictions)
    real_ratings_predictions_df = real_ratings_predictions_df.sort_values(by=['USER_INDEX', 'PREDICTIONS'],
    real_ratings_predictions_df_csv = os.path.join(RESULTS_PATH, F"cluster{cluster_index}_real_ratings_pred
    real_ratings_predictions_df.to_csv(real_ratings_predictions_df_csv)

    real_mse, real_mae = calculate_real_mse_mae(real_ratings, predictions_normalized)
    display(f'real_mean_squared_error={real_mse}, real_mean_absolute_error={real_mae}')

    return train_mse, train_mae, test_mse, test_mae, real_mse, real_mae


# THANOS NEURAL NETWORK
```

```python
display('ratings_normalize_factor', ratings_normalize_factor)

# Create, train and evaluate a Neural Network for each cluster

results: list[list[float]] = []
results_df_index: list[str] = []

for cluster_index in range(len(clusters_ratings)):
    cluster_ratings = clusters_ratings[cluster_index]
    nearest_neighbors = clusters_nearest_neighbors[cluster_index]

    pickle_file = os.path.join(DATAFOLDER_PATH, f"L_K_DEPEND_clusters_nn_original_dfs_{cluster_index}.pkl")
    nn_origin_df = pd.read_pickle(pickle_file)
    nn_filtered_df, nn_filtered_normalized_df = create_nn_filtered_normalized_df(nn_origin_df, NEIGHBOURS_C

    train_mse, train_mae, test_mse, test_mae, real_mse, real_mae = create_train_evaluate_neural_network(
        ratings_normalize_factor, nn_origin_df, nn_filtered_normalized_df, NEIGHBOURS_COLUMNS, cluster_inde
    results.append([train_mse, train_mae, test_mse,
                    test_mae, real_mse, real_mae])
    results_df_index.append(f'CLUSTER_{cluster_index}')
    # Clear memory
    cluster_ratings = nearest_neighbors = pickle_file = nn_origin_df = nn_filtered_df = nn_filtered_normal:


results_df_columns = ['TRAIN_MSE', 'TRAIN_MAE',
                      'TEST_MSE', 'TEST_MAE', 'REAL_MSE', 'REAL_MAE']
results_df = pd.DataFrame(
    results, columns=results_df_columns, index=results_df_index)
display('results_df', results_df)

results_df_csv = os.path.join(RESULTS_PATH, F"results.csv")
results_df.to_csv(results_df_csv, encoding='utf-8')
```

| | | | | | | |
|---|---|---|---|---|---|---|
| **8032150** | 117 | 66322 | 4.0 | 0.0 | 0.0 | 4.0 |
| **8032342** | 117 | 66514 | 10.0 | 0.0 | 0.0 | 10.0 |
| **8032610** | 117 | 66782 | 5.0 | 0.0 | 0.0 | 5.0 |
| **8033783** | 117 | 67955 | 6.0 | 0.0 | 0.0 | 6.0 |

24792 rows × 9 columns

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGHBC |
|---|---|---|---|---|---|---|---|
| **count** | 24792.000000 | 24792.000000 | 24792.000000 | 24792.000000 | 24792.000000 | 24792.000000 | |
| **mean** | 58.297273 | 44131.198330 | 6.598096 | 3.192602 | 2.815545 | 2.102049 | |
| **std** | 33.982039 | 13970.402022 | 2.248538 | 3.715343 | 3.665535 | 3.388091 | |
| **min** | 0.000000 | 17.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 30.000000 | 39645.000000 | 5.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **50%** | 58.000000 | 47816.000000 | 7.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **75%** | 87.000000 | 53325.000000 | 8.000000 | 7.000000 | 7.000000 | 5.000000 | |
| **max** | 117.000000 | 68073.000000 | 10.000000 | 10.000000 | 10.000000 | 10.000000 | |

```
'train_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **1072902** | 15 | 51642 | 9.0 | 0.0 | 9.0 | 8.0 | |
| **1063858** | 15 | 42598 | 8.0 | 7.0 | 8.0 | 0.0 | |
| **7226572** | 106 | 9668 | 9.0 | 0.0 | 0.0 | 0.0 | |
| **1883368** | 27 | 45100 | 4.0 | 4.0 | 0.0 | 0.0 | |
| **2982198** | 43 | 54586 | 3.0 | 0.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **6995823** | 102 | 51255 | 7.0 | 8.0 | 0.0 | 0.0 | |
| **1771390** | 26 | 1206 | 8.0 | 0.0 | 8.0 | 0.0 | |
| **246752** | 3 | 42500 | 2.0 | 2.0 | 0.0 | 0.0 | |
| **5085057** | 74 | 46841 | 8.0 | 0.0 | 0.0 | 0.0 | |
| **7686808** | 112 | 61400 | 9.0 | 0.0 | 7.0 | 0.0 | |

19833 rows × 9 columns

```
'test_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **3926217** | 57 | 45429 | 3.0 | 0.0 | 3.0 | 0.0 | |
| **692868** | 10 | 12028 | 9.0 | 9.0 | 0.0 | 0.0 | |
| **4948031** | 72 | 45983 | 8.0 | 6.0 | 0.0 | 0.0 | |
| **323295** | 4 | 50959 | 8.0 | 8.0 | 0.0 | 0.0 | |
| **4788048** | 70 | 22168 | 8.0 | 0.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **7884497** | 115 | 54837 | 7.0 | 7.0 | 8.0 | 8.0 | |
| **6593821** | 96 | 57757 | 8.0 | 7.0 | 8.0 | 0.0 | |
| **4951735** | 72 | 49687 | 7.0 | 0.0 | 0.0 | 0.0 | |
| **3253610** | 47 | 53662 | 7.0 | 10.0 | 0.0 | 7.0 | |
| **8016468** | 117 | 50640 | 5.0 | 0.0 | 9.0 | 5.0 | |

4959 rows × 9 columns

```
'Training the model with the train_df'
```

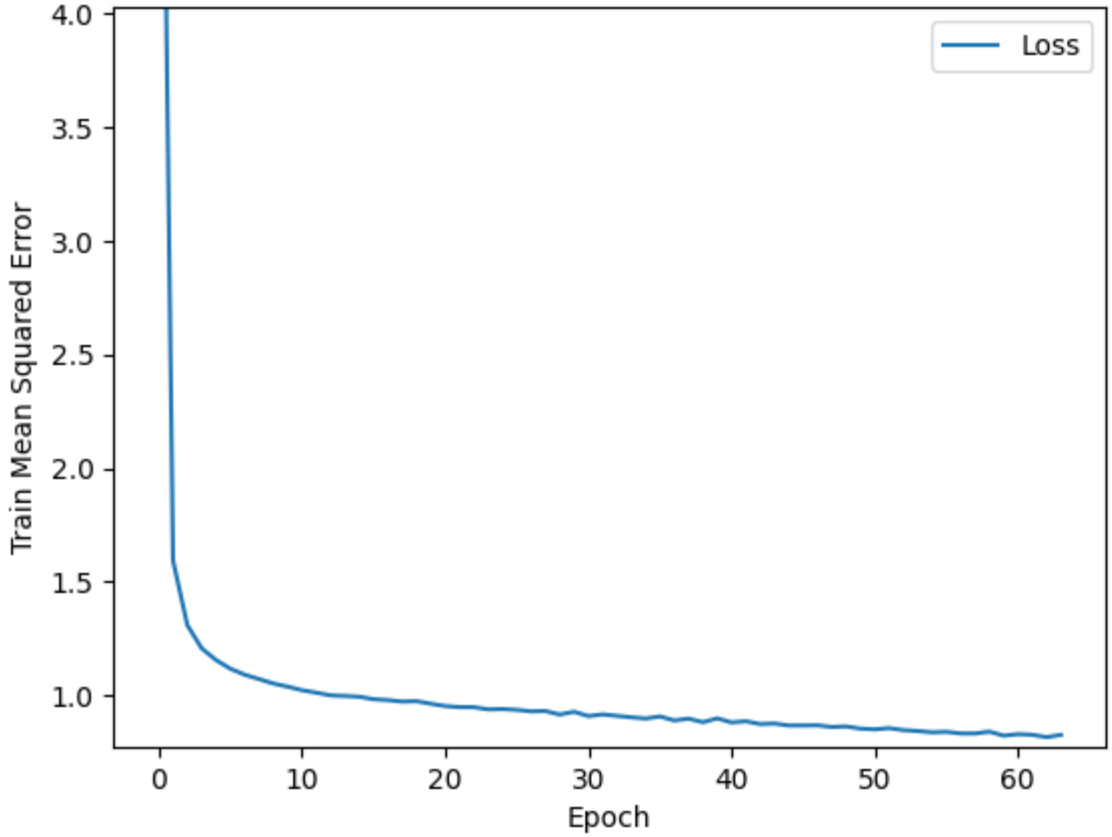
```
Epoch 1/64
155/155 [==============================] - 1s 616us/step - loss: 7.2638 - mean_squared_error: 6.7116 - mea
n_absolute_error: 1.8301
Epoch 2/64
155/155 [==============================] - 0s 597us/step - loss: 2.1100 - mean_squared_error: 1.5931 - mea
n_absolute_error: 0.7929
Epoch 3/64
155/155 [==============================] - 0s 600us/step - loss: 1.8007 - mean_squared_error: 1.3087 - mea
n_absolute_error: 0.6700
Epoch 4/64
155/155 [==============================] - 0s 594us/step - loss: 1.6767 - mean_squared_error: 1.2067 - mea
n_absolute_error: 0.6170
Epoch 5/64
155/155 [==============================] - 0s 598us/step - loss: 1.6058 - mean_squared_error: 1.1555 - mea
n_absolute_error: 0.5823
Epoch 6/64
155/155 [==============================] - 0s 592us/step - loss: 1.5496 - mean_squared_error: 1.1172 - mea
n_absolute_error: 0.5599
Epoch 7/64
155/155 [==============================] - 0s 627us/step - loss: 1.5060 - mean_squared_error: 1.0908 - mea
n_absolute_error: 0.5443
Epoch 8/64
155/155 [==============================] - 0s 616us/step - loss: 1.4714 - mean_squared_error: 1.0717 - mea
n_absolute_error: 0.5394
Epoch 9/64
155/155 [==============================] - 0s 617us/step - loss: 1.4377 - mean_squared_error: 1.0520 - mea
n_absolute_error: 0.5284
Epoch 10/64
155/155 [==============================] - 0s 604us/step - loss: 1.4106 - mean_squared_error: 1.0383 - mea
n_absolute_error: 0.5214
Epoch 11/64
155/155 [==============================] - 0s 591us/step - loss: 1.3826 - mean_squared_error: 1.0226 - mea
n_absolute_error: 0.5155
Epoch 12/64
155/155 [==============================] - 0s 587us/step - loss: 1.3598 - mean_squared_error: 1.0117 - mea
n_absolute_error: 0.5091
Epoch 13/64
155/155 [==============================] - 0s 602us/step - loss: 1.3374 - mean_squared_error: 1.0000 - mea
n_absolute_error: 0.5094
Epoch 14/64
155/155 [==============================] - 0s 740us/step - loss: 1.3245 - mean_squared_error: 0.9972 - mea
n_absolute_error: 0.5074
```

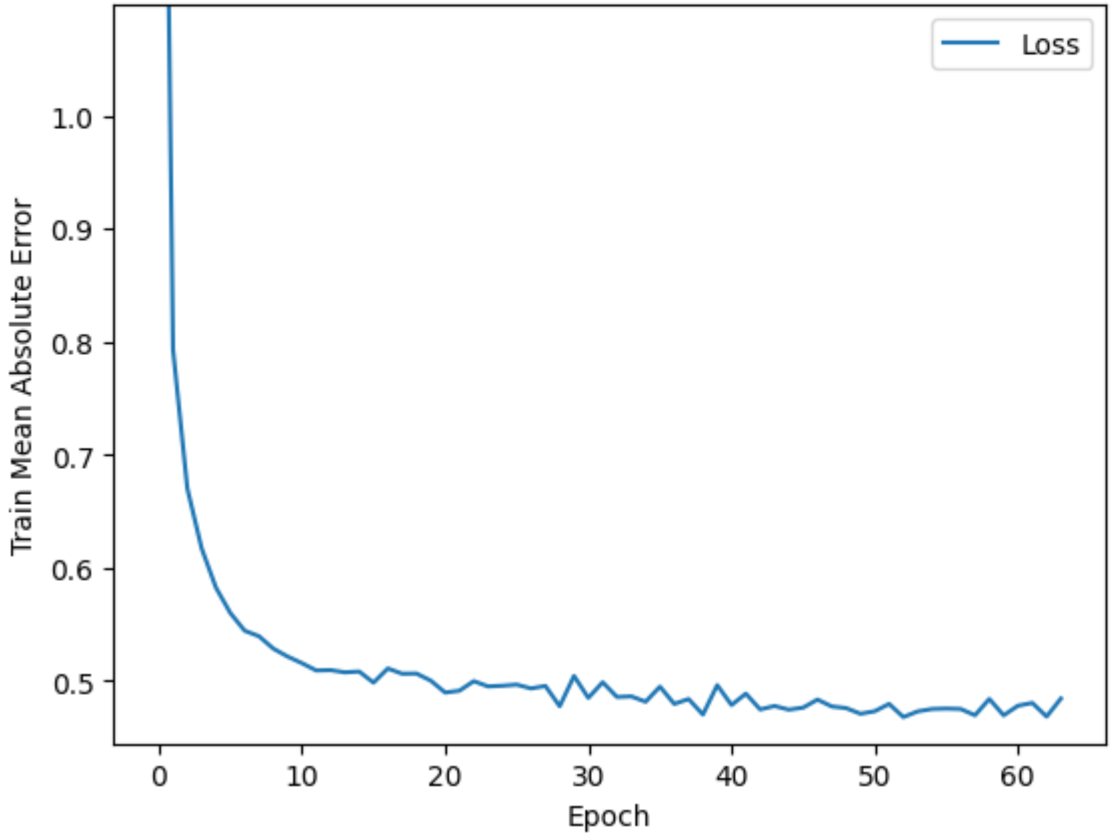
```
Epoch 15/64
155/155 [==============================] - 0s 622us/step - loss: 1.3115 - mean_squared_error: 0.9940 - mea
n_absolute_error: 0.5081
Epoch 16/64
155/155 [==============================] - 0s 587us/step - loss: 1.2918 - mean_squared_error: 0.9830 - mea
n_absolute_error: 0.4983
Epoch 17/64
155/155 [==============================] - 0s 591us/step - loss: 1.2795 - mean_squared_error: 0.9790 - mea
n_absolute_error: 0.5108
Epoch 18/64
155/155 [==============================] - 0s 600us/step - loss: 1.2657 - mean_squared_error: 0.9727 - mea
n_absolute_error: 0.5062
Epoch 19/64
155/155 [==============================] - 0s 628us/step - loss: 1.2602 - mean_squared_error: 0.9743 - mea
n_absolute_error: 0.5063
Epoch 20/64
155/155 [==============================] - 0s 584us/step - loss: 1.2420 - mean_squared_error: 0.9630 - mea
n_absolute_error: 0.5000
Epoch 21/64
155/155 [==============================] - 0s 601us/step - loss: 1.2257 - mean_squared_error: 0.9528 - mea
n_absolute_error: 0.4895
Epoch 22/64
155/155 [==============================] - 0s 602us/step - loss: 1.2153 - mean_squared_error: 0.9483 - mea
n_absolute_error: 0.4912
Epoch 23/64
155/155 [==============================] - 0s 601us/step - loss: 1.2094 - mean_squared_error: 0.9481 - mea
n_absolute_error: 0.4996
Epoch 24/64
155/155 [==============================] - 0s 599us/step - loss: 1.1945 - mean_squared_error: 0.9383 - mea
n_absolute_error: 0.4950
Epoch 25/64
155/155 [==============================] - 0s 630us/step - loss: 1.1913 - mean_squared_error: 0.9396 - mea
n_absolute_error: 0.4956
Epoch 26/64
155/155 [==============================] - 0s 598us/step - loss: 1.1835 - mean_squared_error: 0.9362 - mea
n_absolute_error: 0.4965
Epoch 27/64
155/155 [==============================] - 0s 586us/step - loss: 1.1728 - mean_squared_error: 0.9296 - mea
n_absolute_error: 0.4932
Epoch 28/64
155/155 [==============================] - 0s 597us/step - loss: 1.1703 - mean_squared_error: 0.9309 - mea
n_absolute_error: 0.4954
```

```
Epoch 29/64
155/155 [==============================] - 0s 597us/step - loss: 1.1511 - mean_squared_error: 0.9156 - mea
n_absolute_error: 0.4774
Epoch 30/64
155/155 [==============================] - 0s 585us/step - loss: 1.1596 - mean_squared_error: 0.9276 - mea
n_absolute_error: 0.5042
Epoch 31/64
155/155 [==============================] - 0s 687us/step - loss: 1.1377 - mean_squared_error: 0.9088 - mea
n_absolute_error: 0.4848
Epoch 32/64
155/155 [==============================] - 0s 644us/step - loss: 1.1424 - mean_squared_error: 0.9162 - mea
n_absolute_error: 0.4987
Epoch 33/64
155/155 [==============================] - 0s 600us/step - loss: 1.1331 - mean_squared_error: 0.9097 - mea
n_absolute_error: 0.4856
Epoch 34/64
155/155 [==============================] - 0s 596us/step - loss: 1.1248 - mean_squared_error: 0.9034 - mea
n_absolute_error: 0.4861
Epoch 35/64
155/155 [==============================] - 0s 607us/step - loss: 1.1168 - mean_squared_error: 0.8979 - mea
n_absolute_error: 0.4814
Epoch 36/64
155/155 [==============================] - 0s 592us/step - loss: 1.1236 - mean_squared_error: 0.9070 - mea
n_absolute_error: 0.4948
Epoch 37/64
155/155 [==============================] - 0s 602us/step - loss: 1.1038 - mean_squared_error: 0.8895 - mea
n_absolute_error: 0.4795
Epoch 38/64
155/155 [==============================] - 0s 598us/step - loss: 1.1102 - mean_squared_error: 0.8974 - mea
n_absolute_error: 0.4838
Epoch 39/64
155/155 [==============================] - 0s 601us/step - loss: 1.0929 - mean_squared_error: 0.8817 - mea
n_absolute_error: 0.4701
Epoch 40/64
155/155 [==============================] - 0s 588us/step - loss: 1.1078 - mean_squared_error: 0.8988 - mea
n_absolute_error: 0.4961
Epoch 41/64
155/155 [==============================] - 0s 622us/step - loss: 1.0886 - mean_squared_error: 0.8809 - mea
n_absolute_error: 0.4785
Epoch 42/64
155/155 [==============================] - 0s 600us/step - loss: 1.0916 - mean_squared_error: 0.8860 - mea
n_absolute_error: 0.4886
```

```
Epoch 43/64
155/155 [==============================] - 0s 591us/step - loss: 1.0785 - mean_squared_error: 0.8736 - mea
n_absolute_error: 0.4746
Epoch 44/64
155/155 [==============================] - 0s 590us/step - loss: 1.0797 - mean_squared_error: 0.8761 - mea
n_absolute_error: 0.4776
Epoch 45/64
155/155 [==============================] - 0s 598us/step - loss: 1.0695 - mean_squared_error: 0.8673 - mea
n_absolute_error: 0.4744
Epoch 46/64
155/155 [==============================] - 0s 604us/step - loss: 1.0687 - mean_squared_error: 0.8675 - mea
n_absolute_error: 0.4762
Epoch 47/64
155/155 [==============================] - 0s 591us/step - loss: 1.0683 - mean_squared_error: 0.8682 - mea
n_absolute_error: 0.4833
Epoch 48/64
155/155 [==============================] - 0s 597us/step - loss: 1.0595 - mean_squared_error: 0.8609 - mea
n_absolute_error: 0.4772
Epoch 49/64
155/155 [==============================] - 0s 599us/step - loss: 1.0611 - mean_squared_error: 0.8632 - mea
n_absolute_error: 0.4758
Epoch 50/64
155/155 [==============================] - 0s 597us/step - loss: 1.0503 - mean_squared_error: 0.8536 - mea
n_absolute_error: 0.4708
Epoch 51/64
155/155 [==============================] - 0s 597us/step - loss: 1.0461 - mean_squared_error: 0.8504 - mea
n_absolute_error: 0.4730
Epoch 52/64
155/155 [==============================] - 0s 612us/step - loss: 1.0503 - mean_squared_error: 0.8557 - mea
n_absolute_error: 0.4795
Epoch 53/64
155/155 [==============================] - 0s 587us/step - loss: 1.0405 - mean_squared_error: 0.8470 - mea
n_absolute_error: 0.4679
Epoch 54/64
155/155 [==============================] - 0s 598us/step - loss: 1.0362 - mean_squared_error: 0.8431 - mea
n_absolute_error: 0.4729
Epoch 55/64
155/155 [==============================] - 0s 601us/step - loss: 1.0299 - mean_squared_error: 0.8373 - mea
n_absolute_error: 0.4750
Epoch 56/64
155/155 [==============================] - 0s 612us/step - loss: 1.0310 - mean_squared_error: 0.8396 - mea
n_absolute_error: 0.4753
```

```
Epoch 57/64
155/155 [==============================] - 0s 603us/step - loss: 1.0236 - mean_squared_error: 0.8325 - mea
n_absolute_error: 0.4750
Epoch 58/64
155/155 [==============================] - 0s 594us/step - loss: 1.0223 - mean_squared_error: 0.8321 - mea
n_absolute_error: 0.4694
Epoch 59/64
155/155 [==============================] - 0s 603us/step - loss: 1.0300 - mean_squared_error: 0.8404 - mea
n_absolute_error: 0.4839
Epoch 60/64
155/155 [==============================] - 0s 607us/step - loss: 1.0113 - mean_squared_error: 0.8225 - mea
n_absolute_error: 0.4694
Epoch 61/64
155/155 [==============================] - 0s 597us/step - loss: 1.0172 - mean_squared_error: 0.8293 - mea
n_absolute_error: 0.4779
Epoch 62/64
155/155 [==============================] - 0s 596us/step - loss: 1.0139 - mean_squared_error: 0.8264 - mea
n_absolute_error: 0.4804
Epoch 63/64
155/155 [==============================] - 0s 612us/step - loss: 1.0029 - mean_squared_error: 0.8160 - mea
n_absolute_error: 0.4683
Epoch 64/64
155/155 [==============================] - 0s 595us/step - loss: 1.0119 - mean_squared_error: 0.8260 - mea
n_absolute_error: 0.4843
```


Loss
Train Mean Squared Error
Epoch
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
10
20
30
40
50
60




```
'Evaluating the model against the test_df'
39/39 [==============================] - 0s 594us/step - loss: 1.1000 - mean_squared_error: 0.9149 - mean_
absolute_error: 0.5256
'Predicting the nn_origin_df and comparing with the initial data'
62765/62765 [==============================] - 25s 392us/step
'predictions'
array([1.0582136, 1.0582136, 1.0582136, ..., 1.0582136, 1.0582136,
       1.0582136], dtype=float32)
'predictions.shape'
(8033912,)
'real_ratings'
array([0., 0., 0., ..., 0., 0., 0.])
'real_mean_squared_error=0.5047936831685411, real_mean_absolute_error=0.5860511324230985'
'Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGI |
|---|---|---|---|---|---|---|---|
| **279** | 0 | 279 | 9.0 | 0.0 | 6.0 | 0.0 | |
| **312** | 0 | 312 | 10.0 | 0.0 | 0.0 | 0.0 | |
| **1812** | 0 | 1812 | 7.0 | 0.0 | 10.0 | 0.0 | |
| **4344** | 0 | 4344 | 10.0 | 0.0 | 0.0 | 0.0 | |
| **6020** | 0 | 6020 | 10.0 | 0.0 | 10.0 | 7.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **37922618** | 556 | 67914 | 7.0 | 0.0 | 7.0 | 0.0 | |
| **37922626** | 556 | 67922 | 9.0 | 0.0 | 9.0 | 0.0 | |
| **37922627** | 556 | 67923 | 8.0 | 0.0 | 8.0 | 0.0 | |
| **37922633** | 556 | 67929 | 7.0 | 0.0 | 7.0 | 0.0 | |
| **37922665** | 556 | 67961 | 9.0 | 0.0 | 9.0 | 0.0 | |

112488 rows × 9 columns

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGHE |
|---|---|---|---|---|---|---|---|
| **count** | 112488.000000 | 112488.000000 | 112488.000000 | 112488.000000 | 112488.000000 | 112488.000000 | |
| **mean** | 278.922703 | 32337.910328 | 6.491048 | 2.355113 | 2.298192 | 1.528314 | |
| **std** | 160.942387 | 19622.785983 | 2.574237 | 3.502299 | 3.492612 | 3.013522 | |
| **min** | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 139.000000 | 14491.000000 | 5.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **50%** | 278.000000 | 31301.000000 | 7.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **75%** | 418.000000 | 48995.250000 | 8.000000 | 5.000000 | 5.000000 | 0.000000 | |
| **max** | 556.000000 | 68083.000000 | 10.000000 | 10.000000 | 10.000000 | 10.000000 | |

```
'train_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIG |
|---|---|---|---|---|---|---|---|
| **9373843** | 137 | 46335 | 8.0 | 8.0 | 0.0 | 0.0 | |
| **2048420** | 30 | 5900 | 10.0 | 7.0 | 10.0 | 9.0 | |
| **12351073** | 181 | 27869 | 9.0 | 0.0 | 0.0 | 9.0 | |
| **27305787** | 401 | 4103 | 6.0 | 6.0 | 0.0 | 0.0 | |
| **14741228** | 216 | 35084 | 8.0 | 0.0 | 0.0 | 8.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **25871173** | 379 | 67337 | 3.0 | 0.0 | 3.0 | 0.0 | |
| **37207587** | 546 | 33723 | 10.0 | 10.0 | 0.0 | 0.0 | |
| **35180552** | 516 | 49208 | 1.0 | 1.0 | 0.0 | 0.0 | |
| **420887** | 6 | 12383 | 5.0 | 8.0 | 0.0 | 0.0 | |
| **5330450** | 78 | 19898 | 5.0 | 0.0 | 0.0 | 5.0 | |

89990 rows × 9 columns

```
'test_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIG |
|---|---|---|---|---|---|---|---|
| **28927823** | 424 | 60207 | 4.0 | 4.0 | 0.0 | 0.0 | |
| **15953806** | 234 | 22150 | 9.0 | 0.0 | 9.0 | 0.0 | |
| **18629252** | 273 | 42320 | 1.0 | 0.0 | 0.0 | 0.0 | |
| **33915116** | 498 | 9284 | 8.0 | 8.0 | 0.0 | 0.0 | |
| **16756117** | 246 | 7453 | 10.0 | 0.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **11882976** | 174 | 36360 | 8.0 | 8.0 | 0.0 | 0.0 | |
| **9376113** | 137 | 48605 | 5.0 | 5.0 | 0.0 | 0.0 | |
| **23437217** | 344 | 16321 | 7.0 | 0.0 | 0.0 | 0.0 | |
| **13249288** | 194 | 40992 | 9.0 | 8.0 | 0.0 | 0.0 | |
| **22625098** | 332 | 21210 | 7.0 | 0.0 | 0.0 | 0.0 | |

22498 rows × 9 columns

```
'Training the model with the train_df'
```

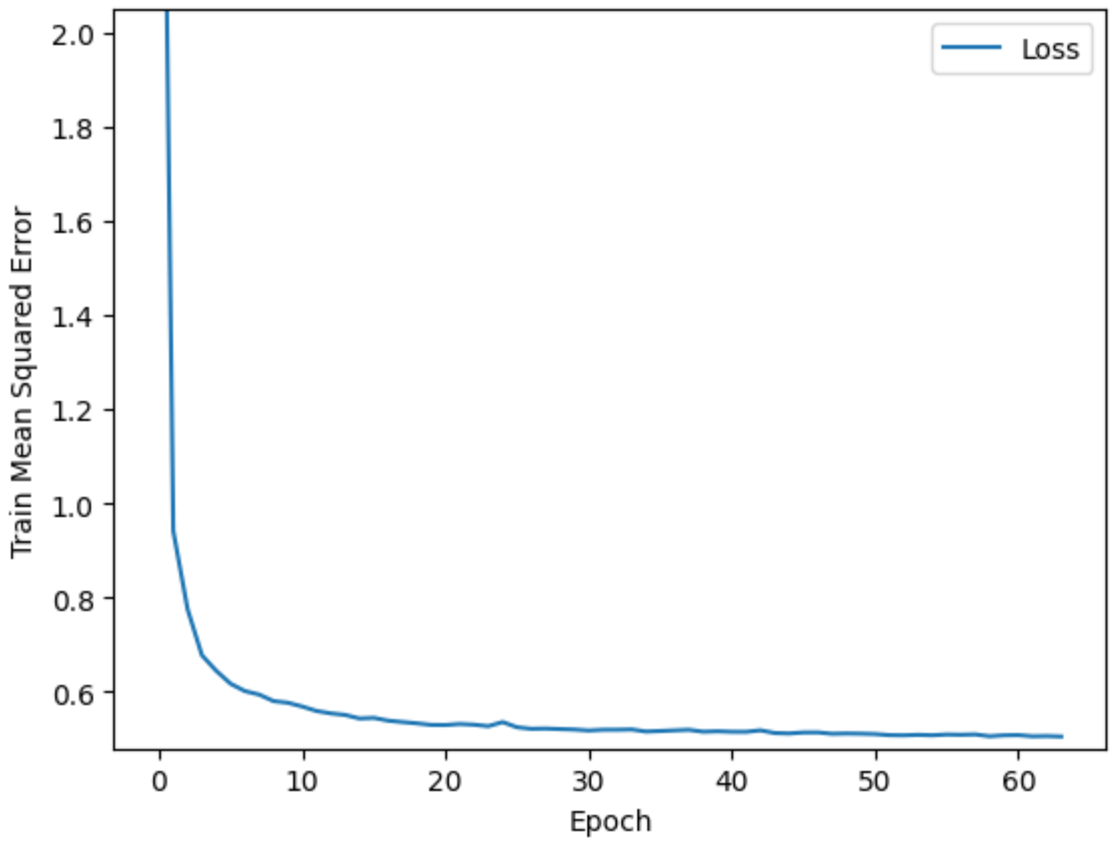
```
Epoch 1/64
704/704 [==============================] - 1s 600us/step - loss: 3.8360 - mean_squared_error: 3.4145 - mea
n_absolute_error: 1.0372
Epoch 2/64
704/704 [==============================] - 0s 589us/step - loss: 1.3065 - mean_squared_error: 0.9415 - mea
n_absolute_error: 0.4499
Epoch 3/64
704/704 [==============================] - 0s 592us/step - loss: 1.1053 - mean_squared_error: 0.7752 - mea
n_absolute_error: 0.3868
Epoch 4/64
704/704 [==============================] - 0s 589us/step - loss: 0.9846 - mean_squared_error: 0.6776 - mea
n_absolute_error: 0.3475
Epoch 5/64
704/704 [==============================] - 0s 588us/step - loss: 0.9287 - mean_squared_error: 0.6447 - mea
n_absolute_error: 0.3342
Epoch 6/64
704/704 [==============================] - 0s 585us/step - loss: 0.8825 - mean_squared_error: 0.6173 - mea
n_absolute_error: 0.3230
Epoch 7/64
704/704 [==============================] - 0s 584us/step - loss: 0.8486 - mean_squared_error: 0.6016 - mea
n_absolute_error: 0.3202
Epoch 8/64
704/704 [==============================] - 0s 592us/step - loss: 0.8247 - mean_squared_error: 0.5942 - mea
n_absolute_error: 0.3227
Epoch 9/64
704/704 [==============================] - 0s 622us/step - loss: 0.7989 - mean_squared_error: 0.5804 - mea
n_absolute_error: 0.3119
Epoch 10/64
704/704 [==============================] - 0s 590us/step - loss: 0.7840 - mean_squared_error: 0.5769 - mea
n_absolute_error: 0.3171
Epoch 11/64
704/704 [==============================] - 0s 592us/step - loss: 0.7673 - mean_squared_error: 0.5689 - mea
n_absolute_error: 0.3150
Epoch 12/64
704/704 [==============================] - 0s 582us/step - loss: 0.7466 - mean_squared_error: 0.5594 - mea
n_absolute_error: 0.3083
Epoch 13/64
704/704 [==============================] - 0s 592us/step - loss: 0.7334 - mean_squared_error: 0.5543 - mea
n_absolute_error: 0.3087
Epoch 14/64
704/704 [==============================] - 0s 586us/step - loss: 0.7229 - mean_squared_error: 0.5510 - mea
n_absolute_error: 0.3110
```

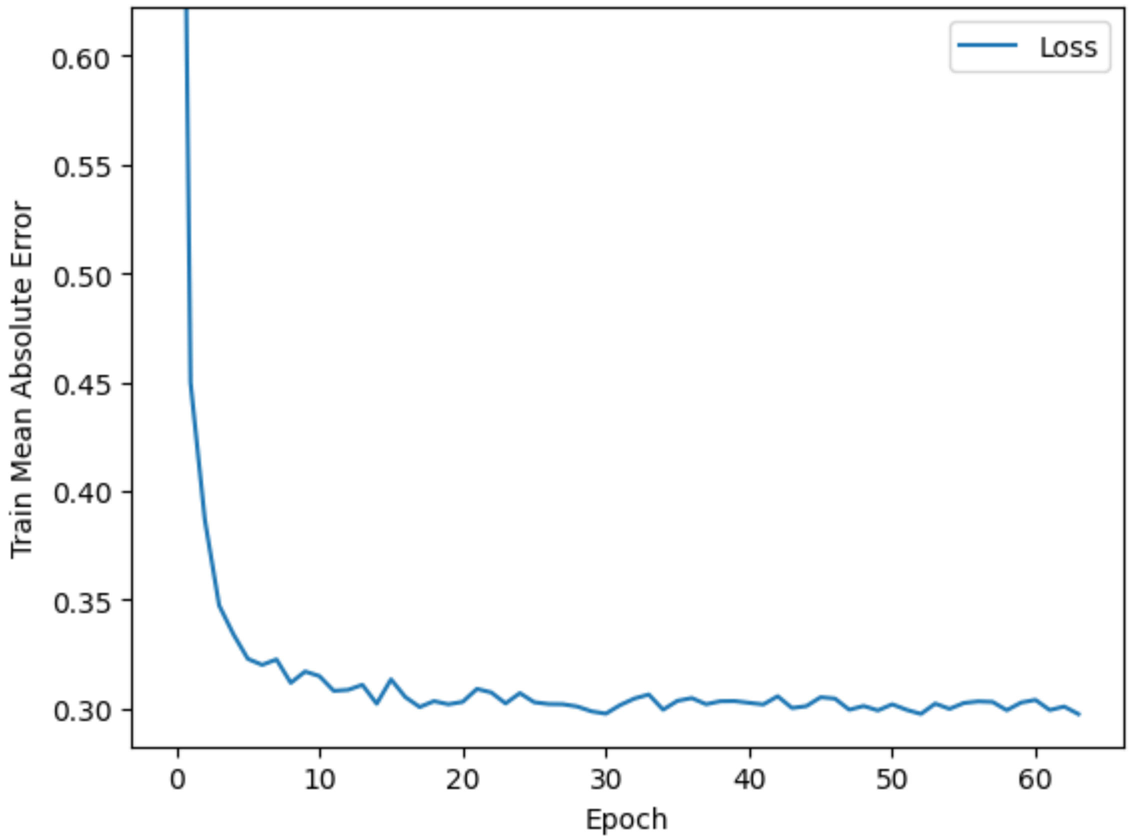
```
Epoch 15/64
704/704 [==============================] - 0s 589us/step - loss: 0.7095 - mean_squared_error: 0.5435 - mea
n_absolute_error: 0.3023
Epoch 16/64
704/704 [==============================] - 0s 594us/step - loss: 0.7062 - mean_squared_error: 0.5447 - mea
n_absolute_error: 0.3135
Epoch 17/64
704/704 [==============================] - 0s 588us/step - loss: 0.6955 - mean_squared_error: 0.5389 - mea
n_absolute_error: 0.3054
Epoch 18/64
704/704 [==============================] - 0s 587us/step - loss: 0.6877 - mean_squared_error: 0.5359 - mea
n_absolute_error: 0.3008
Epoch 19/64
704/704 [==============================] - 0s 588us/step - loss: 0.6811 - mean_squared_error: 0.5332 - mea
n_absolute_error: 0.3035
Epoch 20/64
704/704 [==============================] - 0s 592us/step - loss: 0.6742 - mean_squared_error: 0.5299 - mea
n_absolute_error: 0.3021
Epoch 21/64
704/704 [==============================] - 0s 595us/step - loss: 0.6698 - mean_squared_error: 0.5296 - mea
n_absolute_error: 0.3032
Epoch 22/64
704/704 [==============================] - 0s 615us/step - loss: 0.6688 - mean_squared_error: 0.5318 - mea
n_absolute_error: 0.3091
Epoch 23/64
704/704 [==============================] - 0s 610us/step - loss: 0.6636 - mean_squared_error: 0.5303 - mea
n_absolute_error: 0.3075
Epoch 24/64
704/704 [==============================] - 0s 609us/step - loss: 0.6593 - mean_squared_error: 0.5272 - mea
n_absolute_error: 0.3025
Epoch 25/64
704/704 [==============================] - 0s 599us/step - loss: 0.6663 - mean_squared_error: 0.5359 - mea
n_absolute_error: 0.3072
Epoch 26/64
704/704 [==============================] - 0s 619us/step - loss: 0.6526 - mean_squared_error: 0.5249 - mea
n_absolute_error: 0.3030
Epoch 27/64
704/704 [==============================] - 0s 643us/step - loss: 0.6469 - mean_squared_error: 0.5215 - mea
n_absolute_error: 0.3022
Epoch 28/64
704/704 [==============================] - 0s 635us/step - loss: 0.6443 - mean_squared_error: 0.5221 - mea
n_absolute_error: 0.3021
```

```
Epoch 29/64
704/704 [==============================] - 0s 645us/step - loss: 0.6410 - mean_squared_error: 0.5210 - mea
n_absolute_error: 0.3011
Epoch 30/64
704/704 [==============================] - 0s 617us/step - loss: 0.6382 - mean_squared_error: 0.5200 - mea
n_absolute_error: 0.2988
Epoch 31/64
704/704 [==============================] - 0s 631us/step - loss: 0.6343 - mean_squared_error: 0.5180 - mea
n_absolute_error: 0.2978
Epoch 32/64
704/704 [==============================] - 0s 635us/step - loss: 0.6340 - mean_squared_error: 0.5195 - mea
n_absolute_error: 0.3017
Epoch 33/64
704/704 [==============================] - 0s 602us/step - loss: 0.6327 - mean_squared_error: 0.5196 - mea
n_absolute_error: 0.3047
Epoch 34/64
704/704 [==============================] - 0s 642us/step - loss: 0.6315 - mean_squared_error: 0.5202 - mea
n_absolute_error: 0.3066
Epoch 35/64
704/704 [==============================] - 0s 632us/step - loss: 0.6261 - mean_squared_error: 0.5158 - mea
n_absolute_error: 0.2996
Epoch 36/64
704/704 [==============================] - 0s 638us/step - loss: 0.6266 - mean_squared_error: 0.5169 - mea
n_absolute_error: 0.3037
Epoch 37/64
704/704 [==============================] - 0s 595us/step - loss: 0.6259 - mean_squared_error: 0.5182 - mea
n_absolute_error: 0.3049
Epoch 38/64
704/704 [==============================] - 0s 613us/step - loss: 0.6268 - mean_squared_error: 0.5194 - mea
n_absolute_error: 0.3021
Epoch 39/64
704/704 [==============================] - 0s 590us/step - loss: 0.6217 - mean_squared_error: 0.5154 - mea
n_absolute_error: 0.3035
Epoch 40/64
704/704 [==============================] - 0s 616us/step - loss: 0.6211 - mean_squared_error: 0.5163 - mea
n_absolute_error: 0.3036
Epoch 41/64
704/704 [==============================] - 0s 641us/step - loss: 0.6186 - mean_squared_error: 0.5153 - mea
n_absolute_error: 0.3028
Epoch 42/64
704/704 [==============================] - 0s 628us/step - loss: 0.6182 - mean_squared_error: 0.5152 - mea
n_absolute_error: 0.3020
```

```
Epoch 43/64
704/704 [==============================] - 0s 614us/step - loss: 0.6204 - mean_squared_error: 0.5183 - mea
n_absolute_error: 0.3058
Epoch 44/64
704/704 [==============================] - 0s 585us/step - loss: 0.6140 - mean_squared_error: 0.5126 - mea
n_absolute_error: 0.3004
Epoch 45/64
704/704 [==============================] - 0s 580us/step - loss: 0.6131 - mean_squared_error: 0.5117 - mea
n_absolute_error: 0.3012
Epoch 46/64
704/704 [==============================] - 0s 613us/step - loss: 0.6142 - mean_squared_error: 0.5137 - mea
n_absolute_error: 0.3053
Epoch 47/64
704/704 [==============================] - 0s 620us/step - loss: 0.6131 - mean_squared_error: 0.5139 - mea
n_absolute_error: 0.3047
Epoch 48/64
704/704 [==============================] - 0s 609us/step - loss: 0.6098 - mean_squared_error: 0.5110 - mea
n_absolute_error: 0.2996
Epoch 49/64
704/704 [==============================] - 0s 599us/step - loss: 0.6099 - mean_squared_error: 0.5116 - mea
n_absolute_error: 0.3013
Epoch 50/64
704/704 [==============================] - 0s 610us/step - loss: 0.6086 - mean_squared_error: 0.5112 - mea
n_absolute_error: 0.2993
Epoch 51/64
704/704 [==============================] - 0s 599us/step - loss: 0.6067 - mean_squared_error: 0.5104 - mea
n_absolute_error: 0.3021
Epoch 52/64
704/704 [==============================] - 0s 611us/step - loss: 0.6051 - mean_squared_error: 0.5081 - mea
n_absolute_error: 0.2995
Epoch 53/64
704/704 [==============================] - 0s 595us/step - loss: 0.6029 - mean_squared_error: 0.5078 - mea
n_absolute_error: 0.2976
Epoch 54/64
704/704 [==============================] - 0s 612us/step - loss: 0.6037 - mean_squared_error: 0.5087 - mea
n_absolute_error: 0.3024
Epoch 55/64
704/704 [==============================] - 0s 607us/step - loss: 0.6022 - mean_squared_error: 0.5079 - mea
n_absolute_error: 0.3000
Epoch 56/64
704/704 [==============================] - 0s 612us/step - loss: 0.6030 - mean_squared_error: 0.5094 - mea
n_absolute_error: 0.3027
```

```
Epoch 57/64
704/704 [==============================] - 0s 608us/step - loss: 0.6014 - mean_squared_error: 0.5089 - mea
n_absolute_error: 0.3034
Epoch 58/64
704/704 [==============================] - 0s 607us/step - loss: 0.6022 - mean_squared_error: 0.5096 - mea
n_absolute_error: 0.3032
Epoch 59/64
704/704 [==============================] - 0s 613us/step - loss: 0.5982 - mean_squared_error: 0.5057 - mea
n_absolute_error: 0.2994
Epoch 60/64
704/704 [==============================] - 0s 600us/step - loss: 0.5996 - mean_squared_error: 0.5078 - mea
n_absolute_error: 0.3029
Epoch 61/64
704/704 [==============================] - 0s 587us/step - loss: 0.5996 - mean_squared_error: 0.5082 - mea
n_absolute_error: 0.3041
Epoch 62/64
704/704 [==============================] - 0s 585us/step - loss: 0.5958 - mean_squared_error: 0.5057 - mea
n_absolute_error: 0.2994
Epoch 63/64
704/704 [==============================] - 0s 594us/step - loss: 0.5963 - mean_squared_error: 0.5062 - mea
n_absolute_error: 0.3010
Epoch 64/64
704/704 [==============================] - 0s 592us/step - loss: 0.5947 - mean_squared_error: 0.5051 - mea
n_absolute_error: 0.2976
```


Loss
Train Mean Squared Error
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0
10
20
30
40
50
60
Epoch




```
'Evaluating the model against the test_df'
176/176 [==============================] - 0s 504us/step - loss: 0.6033 - mean_squared_error: 0.5139 - mea
n_absolute_error: 0.3075
'Predicting the nn_origin_df and comparing with the initial data'
     1/296272 [..............................] - ETA: 12:38:40
2023-04-09 12:50:03.445307: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 910146912 e
xceeds 10% of free system memory.
296272/296272 [==============================] - 115s 388us/step
'predictions'
array([0.9871968, 0.9871968, 7.0962753, ..., 0.9871968, 0.9871968,
       0.9871968], dtype=float32)
'predictions.shape'
(37922788,)
'real_ratings'
array([0., 0., 0., ..., 0., 0., 0.])
'real_mean_squared_error=0.3230477744278246, real_mean_absolute_error=0.4385373931867546'
```

'Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating'

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGI |
|---|---|---|---|---|---|---|---|
| **446** | 0 | 446 | 10.0 | 0.0 | 0.0 | 10.0 | |
| **1131** | 0 | 1131 | 10.0 | 0.0 | 0.0 | 10.0 | |
| **1767** | 0 | 1767 | 10.0 | 0.0 | 0.0 | 10.0 | |
| **1853** | 0 | 1853 | 10.0 | 0.0 | 0.0 | 10.0 | |
| **1880** | 0 | 1880 | 10.0 | 0.0 | 0.0 | 10.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **14228337** | 208 | 66865 | 6.0 | 6.0 | 0.0 | 0.0 | |
| **14228340** | 208 | 66868 | 2.0 | 2.0 | 0.0 | 0.0 | |
| **14228482** | 208 | 67010 | 8.0 | 8.0 | 0.0 | 0.0 | |
| **14228522** | 208 | 67050 | 6.0 | 6.0 | 0.0 | 0.0 | |
| **14228694** | 208 | 67222 | 5.0 | 5.0 | 0.0 | 0.0 | |

44402 rows × 9 columns

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGHBC |
|---|---|---|---|---|---|---|---|
| **count** | 44402.000000 | 44402.000000 | 44402.000000 | 44402.000000 | 44402.000000 | 44402.000000 | |
| **mean** | 103.623778 | 24265.409509 | 7.012725 | 3.227062 | 2.490676 | 2.434823 | |
| **std** | 60.326666 | 11663.901321 | 2.478492 | 3.929266 | 3.668950 | 3.682006 | |
| **min** | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 51.000000 | 16053.000000 | 6.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **50%** | 103.000000 | 23732.000000 | 7.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **75%** | 156.000000 | 30045.000000 | 9.000000 | 7.000000 | 6.000000 | 6.000000 | |
| **max** | 208.000000 | 67986.000000 | 10.000000 | 10.000000 | 10.000000 | 10.000000 | |

```
'train_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIG |
|---|---|---|---|---|---|---|---|
| **9475404** | 139 | 11728 | 10.0 | 0.0 | 10.0 | 0.0 | |
| **7709613** | 113 | 16121 | 3.0 | 3.0 | 0.0 | 0.0 | |
| **12965267** | 190 | 29307 | 8.0 | 0.0 | 0.0 | 0.0 | |
| **3509903** | 51 | 37619 | 1.0 | 0.0 | 0.0 | 0.0 | |
| **10770032** | 158 | 12760 | 10.0 | 10.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **1971015** | 28 | 64663 | 1.0 | 0.0 | 0.0 | 0.0 | |
| **3560746** | 52 | 20378 | 4.0 | 0.0 | 10.0 | 9.0 | |
| **12218359** | 179 | 31323 | 7.0 | 7.0 | 0.0 | 0.0 | |
| **313226** | 4 | 40890 | 4.0 | 0.0 | 0.0 | 4.0 | |
| **5108979** | 75 | 2679 | 10.0 | 0.0 | 0.0 | 10.0 | |

35521 rows × 9 columns

```
'test_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIG |
|---|---|---|---|---|---|---|---|
| **10310642** | 151 | 29958 | 6.0 | 0.0 | 0.0 | 0.0 | |
| **11534433** | 169 | 28237 | 8.0 | 8.0 | 8.0 | 9.0 | |
| **7664336** | 112 | 38928 | 6.0 | 6.0 | 0.0 | 8.0 | |
| **13171108** | 193 | 30896 | 7.0 | 0.0 | 7.0 | 0.0 | |
| **5418107** | 79 | 39471 | 10.0 | 0.0 | 0.0 | 10.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **8199750** | 120 | 29670 | 7.0 | 0.0 | 0.0 | 7.0 | |
| **12695451** | 186 | 31827 | 9.0 | 0.0 | 0.0 | 0.0 | |
| **12817797** | 188 | 18005 | 10.0 | 0.0 | 0.0 | 9.0 | |
| **7107413** | 104 | 26677 | 8.0 | 6.0 | 6.0 | 8.0 | |
| **5336795** | 78 | 26243 | 4.0 | 0.0 | 6.0 | 4.0 | |

8881 rows × 9 columns

```
'Training the model with the train_df'
```

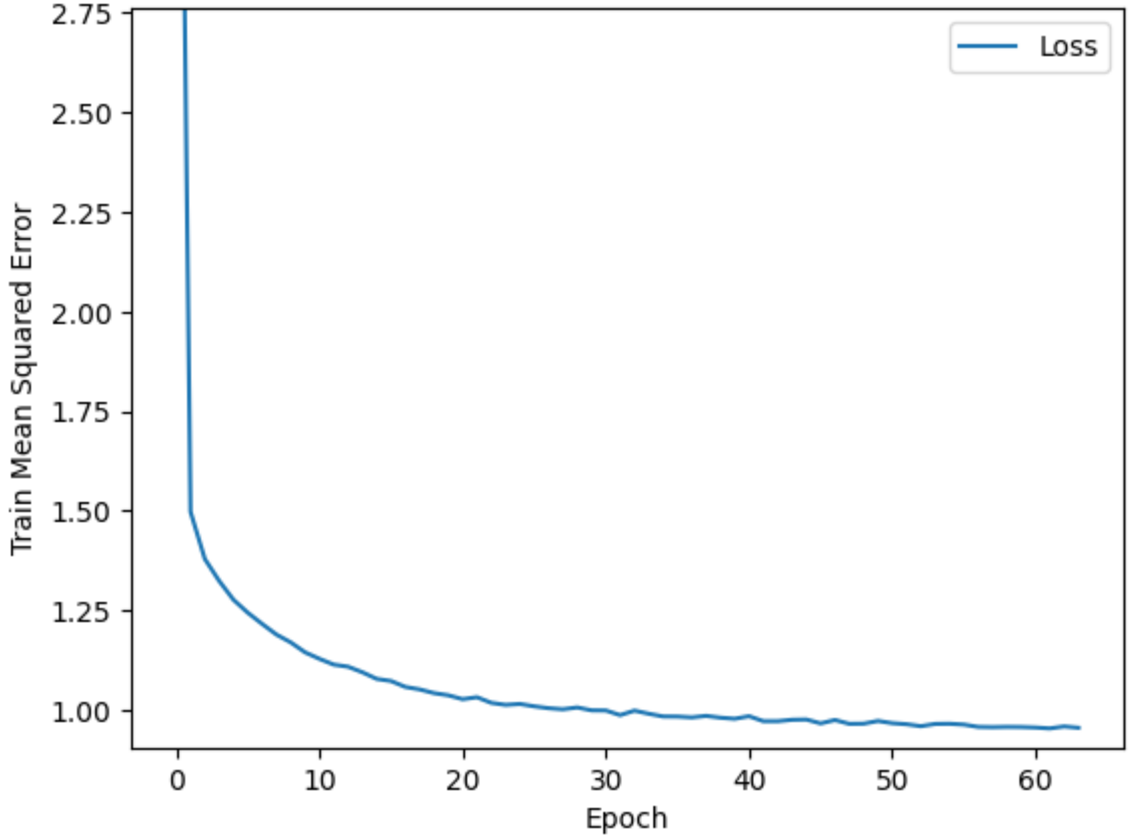
```
Epoch 1/64
278/278 [==============================] - 1s 628us/step - loss: 5.1373 - mean_squared_error: 4.6006 - mea
n_absolute_error: 1.3756
Epoch 2/64
278/278 [==============================] - 0s 609us/step - loss: 1.9941 - mean_squared_error: 1.4974 - mea
n_absolute_error: 0.6796
Epoch 3/64
278/278 [==============================] - 0s 589us/step - loss: 1.8421 - mean_squared_error: 1.3790 - mea
n_absolute_error: 0.6141
Epoch 4/64
278/278 [==============================] - 0s 614us/step - loss: 1.7566 - mean_squared_error: 1.3241 - mea
n_absolute_error: 0.5859
Epoch 5/64
278/278 [==============================] - 0s 599us/step - loss: 1.6816 - mean_squared_error: 1.2768 - mea
n_absolute_error: 0.5695
Epoch 6/64
278/278 [==============================] - 0s 600us/step - loss: 1.6256 - mean_squared_error: 1.2443 - mea
n_absolute_error: 0.5662
Epoch 7/64
278/278 [==============================] - 0s 597us/step - loss: 1.5752 - mean_squared_error: 1.2162 - mea
n_absolute_error: 0.5563
Epoch 8/64
278/278 [==============================] - 0s 598us/step - loss: 1.5288 - mean_squared_error: 1.1895 - mea
n_absolute_error: 0.5456
Epoch 9/64
278/278 [==============================] - 0s 595us/step - loss: 1.4917 - mean_squared_error: 1.1699 - mea
n_absolute_error: 0.5520
Epoch 10/64
278/278 [==============================] - 0s 606us/step - loss: 1.4515 - mean_squared_error: 1.1450 - mea
n_absolute_error: 0.5432
Epoch 11/64
278/278 [==============================] - 0s 598us/step - loss: 1.4230 - mean_squared_error: 1.1287 - mea
n_absolute_error: 0.5359
Epoch 12/64
278/278 [==============================] - 0s 597us/step - loss: 1.3967 - mean_squared_error: 1.1142 - mea
n_absolute_error: 0.5348
Epoch 13/64
278/278 [==============================] - 0s 589us/step - loss: 1.3814 - mean_squared_error: 1.1090 - mea
n_absolute_error: 0.5459
Epoch 14/64
278/278 [==============================] - 0s 595us/step - loss: 1.3575 - mean_squared_error: 1.0950 - mea
n_absolute_error: 0.5325
```

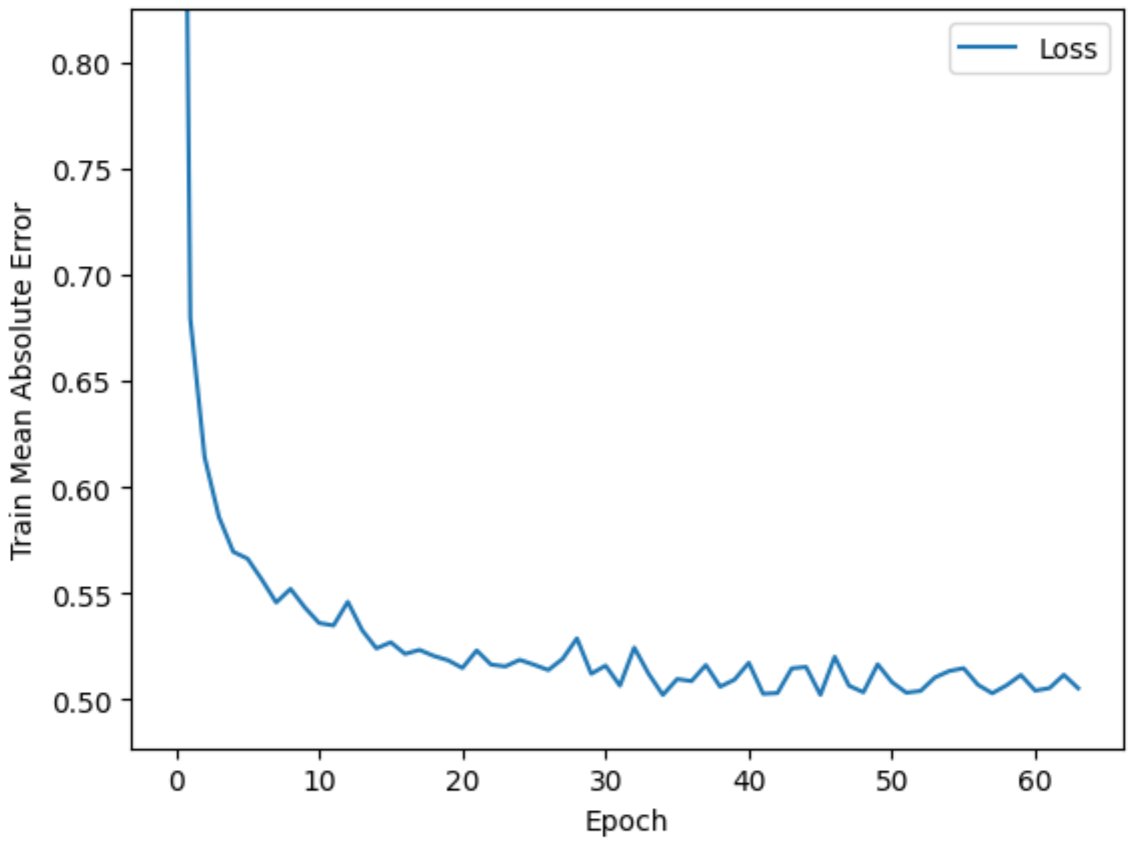
```
Epoch 15/64
278/278 [==============================] - 0s 589us/step - loss: 1.3335 - mean_squared_error: 1.0784 - mea
n_absolute_error: 0.5240
Epoch 16/64
278/278 [==============================] - 0s 592us/step - loss: 1.3208 - mean_squared_error: 1.0731 - mea
n_absolute_error: 0.5268
Epoch 17/64
278/278 [==============================] - 0s 586us/step - loss: 1.3006 - mean_squared_error: 1.0582 - mea
n_absolute_error: 0.5213
Epoch 18/64
278/278 [==============================] - 0s 593us/step - loss: 1.2887 - mean_squared_error: 1.0520 - mea
n_absolute_error: 0.5232
Epoch 19/64
278/278 [==============================] - 0s 666us/step - loss: 1.2744 - mean_squared_error: 1.0427 - mea
n_absolute_error: 0.5204
Epoch 20/64
278/278 [==============================] - 0s 600us/step - loss: 1.2632 - mean_squared_error: 1.0372 - mea
n_absolute_error: 0.5183
Epoch 21/64
278/278 [==============================] - 0s 622us/step - loss: 1.2503 - mean_squared_error: 1.0275 - mea
n_absolute_error: 0.5147
Epoch 22/64
278/278 [==============================] - 0s 596us/step - loss: 1.2509 - mean_squared_error: 1.0324 - mea
n_absolute_error: 0.5230
Epoch 23/64
278/278 [==============================] - 0s 598us/step - loss: 1.2334 - mean_squared_error: 1.0183 - mea
n_absolute_error: 0.5163
Epoch 24/64
278/278 [==============================] - 0s 593us/step - loss: 1.2248 - mean_squared_error: 1.0132 - mea
n_absolute_error: 0.5155
Epoch 25/64
278/278 [==============================] - 0s 598us/step - loss: 1.2230 - mean_squared_error: 1.0155 - mea
n_absolute_error: 0.5185
Epoch 26/64
278/278 [==============================] - 0s 597us/step - loss: 1.2146 - mean_squared_error: 1.0096 - mea
n_absolute_error: 0.5163
Epoch 27/64
278/278 [==============================] - 0s 596us/step - loss: 1.2071 - mean_squared_error: 1.0049 - mea
n_absolute_error: 0.5137
Epoch 28/64
278/278 [==============================] - 0s 607us/step - loss: 1.2019 - mean_squared_error: 1.0022 - mea
n_absolute_error: 0.5190
```

```
Epoch 29/64
278/278 [==============================] - 0s 604us/step - loss: 1.2044 - mean_squared_error: 1.0068 - mea
n_absolute_error: 0.5287
Epoch 30/64
278/278 [==============================] - 0s 595us/step - loss: 1.1943 - mean_squared_error: 0.9995 - mea
n_absolute_error: 0.5120
Epoch 31/64
278/278 [==============================] - 0s 595us/step - loss: 1.1915 - mean_squared_error: 0.9992 - mea
n_absolute_error: 0.5158
Epoch 32/64
278/278 [==============================] - 0s 602us/step - loss: 1.1782 - mean_squared_error: 0.9875 - mea
n_absolute_error: 0.5064
Epoch 33/64
278/278 [==============================] - 0s 599us/step - loss: 1.1877 - mean_squared_error: 0.9988 - mea
n_absolute_error: 0.5243
Epoch 34/64
278/278 [==============================] - 0s 603us/step - loss: 1.1773 - mean_squared_error: 0.9909 - mea
n_absolute_error: 0.5120
Epoch 35/64
278/278 [==============================] - 0s 594us/step - loss: 1.1686 - mean_squared_error: 0.9842 - mea
n_absolute_error: 0.5020
Epoch 36/64
278/278 [==============================] - 0s 608us/step - loss: 1.1663 - mean_squared_error: 0.9839 - mea
n_absolute_error: 0.5095
Epoch 37/64
278/278 [==============================] - 0s 603us/step - loss: 1.1626 - mean_squared_error: 0.9818 - mea
n_absolute_error: 0.5085
Epoch 38/64
278/278 [==============================] - 0s 598us/step - loss: 1.1653 - mean_squared_error: 0.9857 - mea
n_absolute_error: 0.5161
Epoch 39/64
278/278 [==============================] - 0s 581us/step - loss: 1.1580 - mean_squared_error: 0.9811 - mea
n_absolute_error: 0.5059
Epoch 40/64
278/278 [==============================] - 0s 593us/step - loss: 1.1540 - mean_squared_error: 0.9784 - mea
n_absolute_error: 0.5092
Epoch 41/64
278/278 [==============================] - 0s 589us/step - loss: 1.1584 - mean_squared_error: 0.9848 - mea
n_absolute_error: 0.5171
Epoch 42/64
278/278 [==============================] - 0s 603us/step - loss: 1.1449 - mean_squared_error: 0.9723 - mea
n_absolute_error: 0.5026
```

```
Epoch 43/64
278/278 [==============================] - 0s 600us/step - loss: 1.1440 - mean_squared_error: 0.9723 - mea
n_absolute_error: 0.5029
Epoch 44/64
278/278 [==============================] - 0s 590us/step - loss: 1.1458 - mean_squared_error: 0.9759 - mea
n_absolute_error: 0.5145
Epoch 45/64
278/278 [==============================] - 0s 594us/step - loss: 1.1457 - mean_squared_error: 0.9766 - mea
n_absolute_error: 0.5152
Epoch 46/64
278/278 [==============================] - 0s 595us/step - loss: 1.1344 - mean_squared_error: 0.9667 - mea
n_absolute_error: 0.5021
Epoch 47/64
278/278 [==============================] - 0s 591us/step - loss: 1.1414 - mean_squared_error: 0.9753 - mea
n_absolute_error: 0.5200
Epoch 48/64
278/278 [==============================] - 0s 603us/step - loss: 1.1314 - mean_squared_error: 0.9657 - mea
n_absolute_error: 0.5063
Epoch 49/64
278/278 [==============================] - 0s 582us/step - loss: 1.1298 - mean_squared_error: 0.9661 - mea
n_absolute_error: 0.5033
Epoch 50/64
278/278 [==============================] - 0s 584us/step - loss: 1.1357 - mean_squared_error: 0.9729 - mea
n_absolute_error: 0.5165
Epoch 51/64
278/278 [==============================] - 0s 602us/step - loss: 1.1291 - mean_squared_error: 0.9673 - mea
n_absolute_error: 0.5080
Epoch 52/64
278/278 [==============================] - 0s 597us/step - loss: 1.1247 - mean_squared_error: 0.9647 - mea
n_absolute_error: 0.5031
Epoch 53/64
278/278 [==============================] - 0s 604us/step - loss: 1.1192 - mean_squared_error: 0.9598 - mea
n_absolute_error: 0.5039
Epoch 54/64
278/278 [==============================] - 0s 598us/step - loss: 1.1238 - mean_squared_error: 0.9655 - mea
n_absolute_error: 0.5102
Epoch 55/64
278/278 [==============================] - 0s 603us/step - loss: 1.1232 - mean_squared_error: 0.9659 - mea
n_absolute_error: 0.5133
Epoch 56/64
278/278 [==============================] - 0s 621us/step - loss: 1.1200 - mean_squared_error: 0.9638 - mea
n_absolute_error: 0.5146
```

```
Epoch 57/64
278/278 [==============================] - 0s 593us/step - loss: 1.1135 - mean_squared_error: 0.9580 - mea
n_absolute_error: 0.5068
Epoch 58/64
278/278 [==============================] - 0s 595us/step - loss: 1.1115 - mean_squared_error: 0.9573 - mea
n_absolute_error: 0.5028
Epoch 59/64
278/278 [==============================] - 0s 590us/step - loss: 1.1118 - mean_squared_error: 0.9580 - mea
n_absolute_error: 0.5065
Epoch 60/64
278/278 [==============================] - 0s 591us/step - loss: 1.1105 - mean_squared_error: 0.9577 - mea
n_absolute_error: 0.5114
Epoch 61/64
278/278 [==============================] - 0s 588us/step - loss: 1.1081 - mean_squared_error: 0.9565 - mea
n_absolute_error: 0.5041
Epoch 62/64
278/278 [==============================] - 0s 601us/step - loss: 1.1058 - mean_squared_error: 0.9542 - mea
n_absolute_error: 0.5052
Epoch 63/64
278/278 [==============================] - 0s 609us/step - loss: 1.1092 - mean_squared_error: 0.9591 - mea
n_absolute_error: 0.5115
Epoch 64/64
278/278 [==============================] - 0s 599us/step - loss: 1.1045 - mean_squared_error: 0.9558 - mea
n_absolute_error: 0.5051
```


2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
Train Mean Squared Error
0
10
20
30
40
50
60
Epoch
Loss




```
'Evaluating the model against the test_df'
70/70 [==============================] - 0s 530us/step - loss: 1.1282 - mean_squared_error: 0.9792 - mean_
absolute_error: 0.5001
'Predicting the nn_origin_df and comparing with the initial data'
111169/111169 [==============================] - 43s 383us/step
'predictions'
array([1.0290031, 1.0290031, 1.0290031, ..., 1.0290031, 1.0290031,
       1.0290031], dtype=float32)
'predictions.shape'
(14229556,)
'real_ratings'
array([0., 0., 0., ..., 0., 0., 0.])
'real_mean_squared_error=0.4958635547986294, real_mean_absolute_error=0.5520919238025163'
'Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **16576** | 0 | 16576 | 5.0 | 5.0 | 0.0 | 0.0 | |
| **19891** | 0 | 19891 | 1.0 | 1.0 | 8.0 | 8.0 | |
| **22049** | 0 | 22049 | 10.0 | 10.0 | 0.0 | 0.0 | |
| **23000** | 0 | 23000 | 2.0 | 2.0 | 0.0 | 0.0 | |
| **23179** | 0 | 23179 | 4.0 | 4.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **6804722** | 99 | 64406 | 7.0 | 7.0 | 0.0 | 0.0 | |
| **6804858** | 99 | 64542 | 7.0 | 7.0 | 0.0 | 0.0 | |
| **6805460** | 99 | 65144 | 4.0 | 4.0 | 0.0 | 0.0 | |
| **6807289** | 99 | 66973 | 1.0 | 1.0 | 0.0 | 0.0 | |
| **6807447** | 99 | 67131 | 5.0 | 5.0 | 0.0 | 0.0 | |

21699 rows × 9 columns

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGHBC |
|---|---|---|---|---|---|---|---|
| **count** | 21699.000000 | 21699.000000 | 21699.000000 | 21699.000000 | 21699.000000 | 21699.000000 | |
| **mean** | 48.947970 | 32141.397115 | 7.219641 | 3.610627 | 3.124061 | 3.231577 | |
| **std** | 29.326079 | 15343.412073 | 2.220983 | 3.988919 | 3.937279 | 4.013959 | |
| **min** | 0.000000 | 7.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 23.000000 | 19054.000000 | 6.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **50%** | 50.000000 | 31398.000000 | 8.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **75%** | 74.000000 | 43970.000000 | 9.000000 | 8.000000 | 8.000000 | 8.000000 | |
| **max** | 99.000000 | 68027.000000 | 10.000000 | 10.000000 | 10.000000 | 10.000000 | |

```
'train_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **5486476** | 80 | 39756 | 7.0 | 7.0 | 0.0 | 0.0 | |
| **153997** | 2 | 17829 | 1.0 | 1.0 | 0.0 | 0.0 | |
| **2614275** | 38 | 27083 | 7.0 | 7.0 | 0.0 | 0.0 | |
| **586700** | 8 | 42028 | 6.0 | 6.0 | 6.0 | 0.0 | |
| **3193065** | 46 | 61201 | 5.0 | 5.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **3771980** | 55 | 27360 | 8.0 | 0.0 | 0.0 | 0.0 | |
| **6767048** | 99 | 26732 | 9.0 | 9.0 | 0.0 | 0.0 | |
| **1595376** | 23 | 29444 | 10.0 | 10.0 | 10.0 | 0.0 | |
| **235476** | 3 | 31224 | 9.0 | 8.0 | 0.0 | 9.0 | |
| **4945678** | 72 | 43630 | 8.0 | 0.0 | 8.0 | 0.0 | |

17359 rows × 9 columns

```
'test_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **1810277** | 26 | 40093 | 7.0 | 0.0 | 9.0 | 0.0 | |
| **2769361** | 40 | 46001 | 7.0 | 0.0 | 7.0 | 8.0 | |
| **4404177** | 64 | 46801 | 6.0 | 6.0 | 7.0 | 0.0 | |
| **324073** | 4 | 51737 | 1.0 | 0.0 | 1.0 | 7.0 | |
| **703627** | 10 | 22787 | 7.0 | 0.0 | 7.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **3448056** | 50 | 43856 | 4.0 | 0.0 | 0.0 | 0.0 | |
| **3893079** | 57 | 12291 | 6.0 | 0.0 | 0.0 | 6.0 | |
| **3245049** | 47 | 45101 | 10.0 | 0.0 | 10.0 | 0.0 | |
| **3447837** | 50 | 43637 | 5.0 | 0.0 | 0.0 | 3.0 | |
| **4962615** | 72 | 60567 | 9.0 | 0.0 | 9.0 | 0.0 | |

4340 rows × 9 columns

```
'Training the model with the train_df'
```

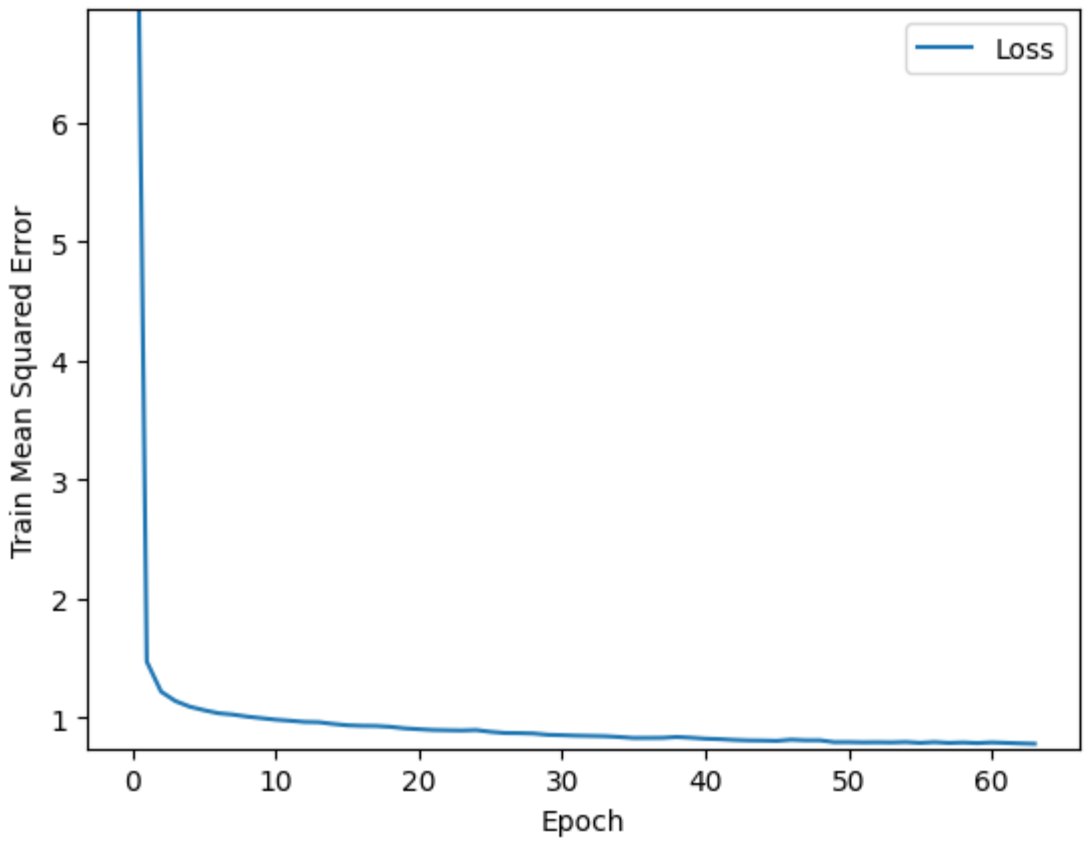
```
Epoch 1/64
136/136 [==============================] - 1s 643us/step - loss: 12.1053 - mean_squared_error: 11.5821 - m
ean_absolute_error: 2.6115
Epoch 2/64
136/136 [==============================] - 0s 602us/step - loss: 1.9703 - mean_squared_error: 1.4729 - mea
n_absolute_error: 0.7831
Epoch 3/64
136/136 [==============================] - 0s 609us/step - loss: 1.6915 - mean_squared_error: 1.2208 - mea
n_absolute_error: 0.6606
Epoch 4/64
136/136 [==============================] - 0s 601us/step - loss: 1.5911 - mean_squared_error: 1.1418 - mea
n_absolute_error: 0.6188
Epoch 5/64
136/136 [==============================] - 0s 613us/step - loss: 1.5260 - mean_squared_error: 1.0946 - mea
n_absolute_error: 0.5842
Epoch 6/64
136/136 [==============================] - 0s 622us/step - loss: 1.4809 - mean_squared_error: 1.0651 - mea
n_absolute_error: 0.5609
Epoch 7/64
136/136 [==============================] - 0s 602us/step - loss: 1.4416 - mean_squared_error: 1.0397 - mea
n_absolute_error: 0.5429
Epoch 8/64
136/136 [==============================] - 0s 629us/step - loss: 1.4171 - mean_squared_error: 1.0286 - mea
n_absolute_error: 0.5385
Epoch 9/64
136/136 [==============================] - 0s 598us/step - loss: 1.3879 - mean_squared_error: 1.0114 - mea
n_absolute_error: 0.5322
Epoch 10/64
136/136 [==============================] - 0s 612us/step - loss: 1.3635 - mean_squared_error: 0.9984 - mea
n_absolute_error: 0.5263
Epoch 11/64
136/136 [==============================] - 0s 619us/step - loss: 1.3400 - mean_squared_error: 0.9855 - mea
n_absolute_error: 0.5173
Epoch 12/64
136/136 [==============================] - 0s 594us/step - loss: 1.3219 - mean_squared_error: 0.9768 - mea
n_absolute_error: 0.5185
Epoch 13/64
136/136 [==============================] - 0s 605us/step - loss: 1.3028 - mean_squared_error: 0.9668 - mea
n_absolute_error: 0.5177
Epoch 14/64
136/136 [==============================] - 0s 596us/step - loss: 1.2914 - mean_squared_error: 0.9643 - mea
n_absolute_error: 0.5246
```

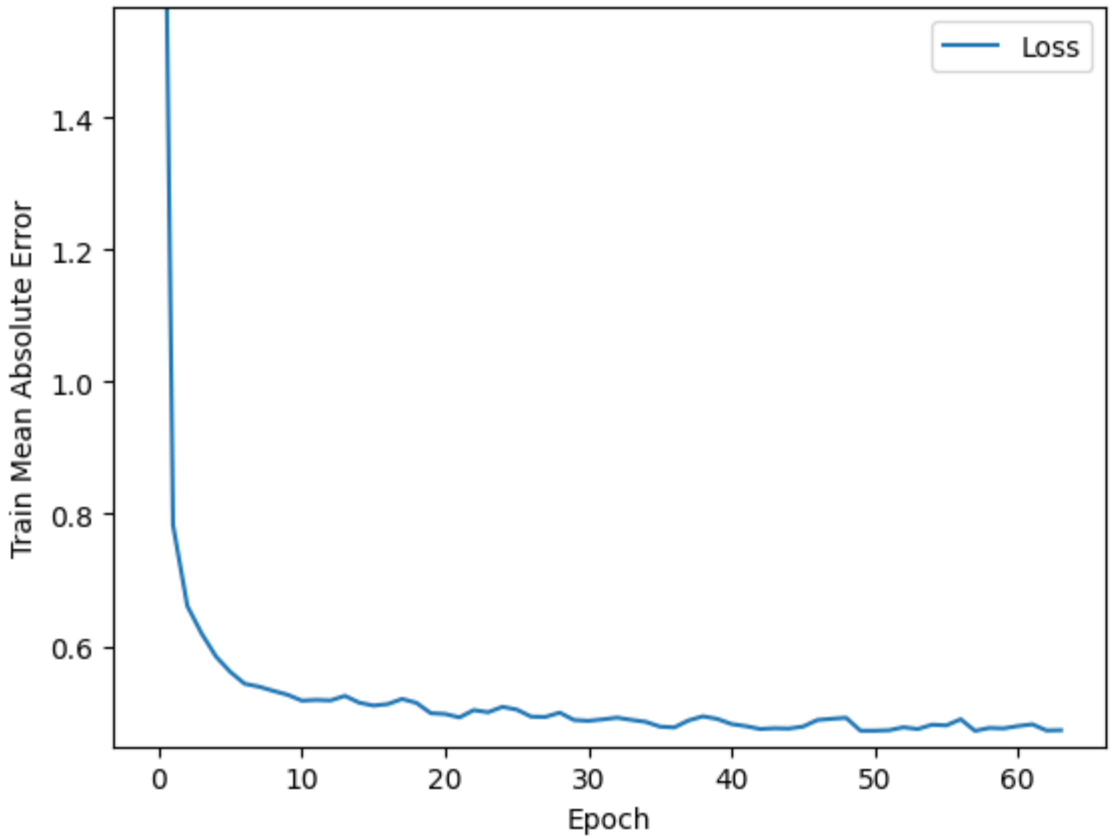
```
Epoch 15/64
136/136 [==============================] - 0s 592us/step - loss: 1.2695 - mean_squared_error: 0.9500 - mea
n_absolute_error: 0.5143
Epoch 16/64
136/136 [==============================] - 0s 606us/step - loss: 1.2503 - mean_squared_error: 0.9385 - mea
n_absolute_error: 0.5099
Epoch 17/64
136/136 [==============================] - 0s 603us/step - loss: 1.2388 - mean_squared_error: 0.9342 - mea
n_absolute_error: 0.5122
Epoch 18/64
136/136 [==============================] - 0s 640us/step - loss: 1.2304 - mean_squared_error: 0.9328 - mea
n_absolute_error: 0.5199
Epoch 19/64
136/136 [==============================] - 0s 610us/step - loss: 1.2170 - mean_squared_error: 0.9257 - mea
n_absolute_error: 0.5140
Epoch 20/64
136/136 [==============================] - 0s 604us/step - loss: 1.1984 - mean_squared_error: 0.9128 - mea
n_absolute_error: 0.4984
Epoch 21/64
136/136 [==============================] - 0s 631us/step - loss: 1.1853 - mean_squared_error: 0.9055 - mea
n_absolute_error: 0.4972
Epoch 22/64
136/136 [==============================] - 0s 596us/step - loss: 1.1736 - mean_squared_error: 0.8990 - mea
n_absolute_error: 0.4920
Epoch 23/64
136/136 [==============================] - 0s 608us/step - loss: 1.1665 - mean_squared_error: 0.8967 - mea
n_absolute_error: 0.5031
Epoch 24/64
136/136 [==============================] - 0s 666us/step - loss: 1.1597 - mean_squared_error: 0.8948 - mea
n_absolute_error: 0.4998
Epoch 25/64
136/136 [==============================] - 0s 693us/step - loss: 1.1590 - mean_squared_error: 0.8983 - mea
n_absolute_error: 0.5080
Epoch 26/64
136/136 [==============================] - 0s 613us/step - loss: 1.1403 - mean_squared_error: 0.8836 - mea
n_absolute_error: 0.5038
Epoch 27/64
136/136 [==============================] - 0s 594us/step - loss: 1.1260 - mean_squared_error: 0.8738 - mea
n_absolute_error: 0.4932
Epoch 28/64
136/136 [==============================] - 0s 589us/step - loss: 1.1205 - mean_squared_error: 0.8727 - mea
n_absolute_error: 0.4927
```

```
Epoch 29/64
136/136 [==============================] - 0s 642us/step - loss: 1.1136 - mean_squared_error: 0.8697 - mea
n_absolute_error: 0.4992
Epoch 30/64
136/136 [==============================] - 0s 596us/step - loss: 1.0988 - mean_squared_error: 0.8587 - mea
n_absolute_error: 0.4881
Epoch 31/64
136/136 [==============================] - 0s 605us/step - loss: 1.0916 - mean_squared_error: 0.8553 - mea
n_absolute_error: 0.4868
Epoch 32/64
136/136 [==============================] - 0s 601us/step - loss: 1.0834 - mean_squared_error: 0.8514 - mea
n_absolute_error: 0.4890
Epoch 33/64
136/136 [==============================] - 0s 607us/step - loss: 1.0767 - mean_squared_error: 0.8492 - mea
n_absolute_error: 0.4916
Epoch 34/64
136/136 [==============================] - 0s 592us/step - loss: 1.0716 - mean_squared_error: 0.8467 - mea
n_absolute_error: 0.4883
Epoch 35/64
136/136 [==============================] - 0s 590us/step - loss: 1.0617 - mean_squared_error: 0.8401 - mea
n_absolute_error: 0.4853
Epoch 36/64
136/136 [==============================] - 0s 590us/step - loss: 1.0497 - mean_squared_error: 0.8318 - mea
n_absolute_error: 0.4781
Epoch 37/64
136/136 [==============================] - 0s 588us/step - loss: 1.0470 - mean_squared_error: 0.8323 - mea
n_absolute_error: 0.4768
Epoch 38/64
136/136 [==============================] - 0s 613us/step - loss: 1.0451 - mean_squared_error: 0.8330 - mea
n_absolute_error: 0.4874
Epoch 39/64
136/136 [==============================] - 0s 634us/step - loss: 1.0491 - mean_squared_error: 0.8404 - mea
n_absolute_error: 0.4937
Epoch 40/64
136/136 [==============================] - 0s 604us/step - loss: 1.0405 - mean_squared_error: 0.8341 - mea
n_absolute_error: 0.4897
Epoch 41/64
136/136 [==============================] - 0s 602us/step - loss: 1.0299 - mean_squared_error: 0.8260 - mea
n_absolute_error: 0.4819
Epoch 42/64
136/136 [==============================] - 0s 617us/step - loss: 1.0235 - mean_squared_error: 0.8220 - mea
n_absolute_error: 0.4788
```

```
Epoch 43/64
136/136 [==============================] - 0s 599us/step - loss: 1.0143 - mean_squared_error: 0.8155 - mea
n_absolute_error: 0.4744
Epoch 44/64
136/136 [==============================] - 0s 597us/step - loss: 1.0084 - mean_squared_error: 0.8113 - mea
n_absolute_error: 0.4756
Epoch 45/64
136/136 [==============================] - 0s 593us/step - loss: 1.0045 - mean_squared_error: 0.8101 - mea
n_absolute_error: 0.4751
Epoch 46/64
136/136 [==============================] - 0s 597us/step - loss: 0.9990 - mean_squared_error: 0.8075 - mea
n_absolute_error: 0.4782
Epoch 47/64
136/136 [==============================] - 0s 599us/step - loss: 1.0073 - mean_squared_error: 0.8174 - mea
n_absolute_error: 0.4881
Epoch 48/64
136/136 [==============================] - 0s 609us/step - loss: 0.9998 - mean_squared_error: 0.8122 - mea
n_absolute_error: 0.4899
Epoch 49/64
136/136 [==============================] - 0s 597us/step - loss: 0.9984 - mean_squared_error: 0.8123 - mea
n_absolute_error: 0.4915
Epoch 50/64
136/136 [==============================] - 0s 600us/step - loss: 0.9810 - mean_squared_error: 0.7968 - mea
n_absolute_error: 0.4719
Epoch 51/64
136/136 [==============================] - 0s 603us/step - loss: 0.9796 - mean_squared_error: 0.7974 - mea
n_absolute_error: 0.4719
Epoch 52/64
136/136 [==============================] - 0s 590us/step - loss: 0.9755 - mean_squared_error: 0.7950 - mea
n_absolute_error: 0.4726
Epoch 53/64
136/136 [==============================] - 0s 588us/step - loss: 0.9751 - mean_squared_error: 0.7960 - mea
n_absolute_error: 0.4771
Epoch 54/64
136/136 [==============================] - 0s 597us/step - loss: 0.9709 - mean_squared_error: 0.7944 - mea
n_absolute_error: 0.4744
Epoch 55/64
136/136 [==============================] - 0s 606us/step - loss: 0.9729 - mean_squared_error: 0.7979 - mea
n_absolute_error: 0.4809
Epoch 56/64
136/136 [==============================] - 0s 632us/step - loss: 0.9646 - mean_squared_error: 0.7907 - mea
n_absolute_error: 0.4799
```

```
Epoch 57/64
136/136 [==============================] - 0s 601us/step - loss: 0.9692 - mean_squared_error: 0.7971 - mea
n_absolute_error: 0.4892
Epoch 58/64
136/136 [==============================] - 0s 596us/step - loss: 0.9606 - mean_squared_error: 0.7902 - mea
n_absolute_error: 0.4716
Epoch 59/64
136/136 [==============================] - 0s 603us/step - loss: 0.9625 - mean_squared_error: 0.7938 - mea
n_absolute_error: 0.4761
Epoch 60/64
136/136 [==============================] - 0s 598us/step - loss: 0.9564 - mean_squared_error: 0.7890 - mea
n_absolute_error: 0.4754
Epoch 61/64
136/136 [==============================] - 0s 603us/step - loss: 0.9602 - mean_squared_error: 0.7944 - mea
n_absolute_error: 0.4789
Epoch 62/64
136/136 [==============================] - 0s 603us/step - loss: 0.9561 - mean_squared_error: 0.7906 - mea
n_absolute_error: 0.4816
Epoch 63/64
136/136 [==============================] - 0s 610us/step - loss: 0.9497 - mean_squared_error: 0.7864 - mea
n_absolute_error: 0.4722
Epoch 64/64
136/136 [==============================] - 0s 593us/step - loss: 0.9459 - mean_squared_error: 0.7832 - mea
n_absolute_error: 0.4727
```


Loss
Train Mean Squared Error
1
2
3
4
5
6
0
10
20
30
40
50
60
Epoch




```
'Evaluating the model against the test_df'
34/34 [==============================] - 0s 559us/step - loss: 0.9118 - mean_squared_error: 0.7500 - mean_
absolute_error: 0.4902
'Predicting the nn_origin_df and comparing with the initial data'
53191/53191 [==============================] - 21s 387us/step
'predictions'
array([1.1999062, 1.1999062, 1.1999062, ..., 1.1999062, 1.1999062,
       1.1999062], dtype=float32)
'predictions.shape'
(6808400,)
'real_ratings'
array([0., 0., 0., ..., 0., 0., 0.])
'real_mean_squared_error=0.48460369867368513, real_mean_absolute_error=0.5719587269698009'
'Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **27825** | 0 | 27825 | 5.0 | 0.0 | 0.0 | 5.0 | |
| **28744** | 0 | 28744 | 5.0 | 8.0 | 0.0 | 5.0 | |
| **38395** | 0 | 38395 | 6.0 | 0.0 | 6.0 | 6.0 | |
| **40242** | 0 | 40242 | 4.0 | 8.0 | 0.0 | 4.0 | |
| **40398** | 0 | 40398 | 4.0 | 8.0 | 7.0 | 4.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **6466517** | 94 | 66621 | 2.0 | 2.0 | 0.0 | 0.0 | |
| **6466678** | 94 | 66782 | 2.0 | 2.0 | 0.0 | 0.0 | |
| **6466705** | 94 | 66809 | 3.0 | 3.0 | 0.0 | 0.0 | |
| **6466730** | 94 | 66834 | 1.0 | 1.0 | 0.0 | 0.0 | |
| **6467272** | 94 | 67376 | 3.0 | 3.0 | 0.0 | 0.0 | |

20181 rows × 9 columns

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGHBO |
|---|---|---|---|---|---|---|---|
| **count** | 20181.000000 | 20181.000000 | 20181.000000 | 20181.000000 | 20181.000000 | 20181.000000 | |
| **mean** | 46.669243 | 49255.189237 | 6.488876 | 3.634062 | 2.803627 | 2.983351 | |
| **std** | 27.661793 | 15667.097322 | 2.331994 | 3.719132 | 3.583817 | 3.694198 | |
| **min** | 0.000000 | 7.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 22.000000 | 43592.000000 | 5.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **50%** | 47.000000 | 54755.000000 | 7.000000 | 3.000000 | 0.000000 | 0.000000 | |
| **75%** | 71.000000 | 60706.000000 | 8.000000 | 7.000000 | 7.000000 | 7.000000 | |
| **max** | 94.000000 | 68055.000000 | 10.000000 | 10.000000 | 10.000000 | 10.000000 | |

```
'train_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **117771** | 1 | 49687 | 7.0 | 0.0 | 0.0 | 7.0 | |
| **1957184** | 28 | 50832 | 8.0 | 8.0 | 10.0 | 0.0 | |
| **3694891** | 54 | 18355 | 3.0 | 0.0 | 0.0 | 0.0 | |
| **1422676** | 20 | 60996 | 10.0 | 8.0 | 10.0 | 10.0 | |
| **3989197** | 58 | 40325 | 9.0 | 0.0 | 9.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **3605173** | 52 | 64805 | 1.0 | 0.0 | 1.0 | 0.0 | |
| **3919183** | 57 | 38395 | 6.0 | 7.0 | 0.0 | 6.0 | |
| **1627639** | 23 | 61707 | 8.0 | 8.0 | 0.0 | 7.0 | |
| **292929** | 4 | 20593 | 7.0 | 0.0 | 0.0 | 7.0 | |
| **5092426** | 74 | 54210 | 8.0 | 5.0 | 8.0 | 5.0 | |

16144 rows × 9 columns

```
'test_df'
```

| | USER_INDEX | MOVIE_INDEX | USER_RATINGS | NEIGHBOR_RATINGS_0 | NEIGHBOR_RATINGS_1 | NEIGHBOR_RATINGS_2 | NEIGH |
|---|---|---|---|---|---|---|---|
| **1972631** | 28 | 66279 | 3.0 | 3.0 | 2.0 | 4.0 | |
| **6102498** | 89 | 43022 | 6.0 | 0.0 | 6.0 | 0.0 | |
| **1866093** | 27 | 27825 | 7.0 | 0.0 | 7.0 | 0.0 | |
| **464311** | 6 | 55807 | 7.0 | 6.0 | 7.0 | 6.0 | |
| **4746590** | 69 | 48794 | 7.0 | 7.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **5407336** | 79 | 28700 | 7.0 | 0.0 | 7.0 | 0.0 | |
| **3386640** | 49 | 50524 | 10.0 | 10.0 | 0.0 | 9.0 | |
| **4966238** | 72 | 64190 | 5.0 | 0.0 | 0.0 | 5.0 | |
| **4850699** | 71 | 16735 | 9.0 | 9.0 | 0.0 | 0.0 | |
| **737861** | 10 | 57021 | 5.0 | 0.0 | 5.0 | 0.0 | |

4037 rows × 9 columns

```
'Training the model with the train_df'
```

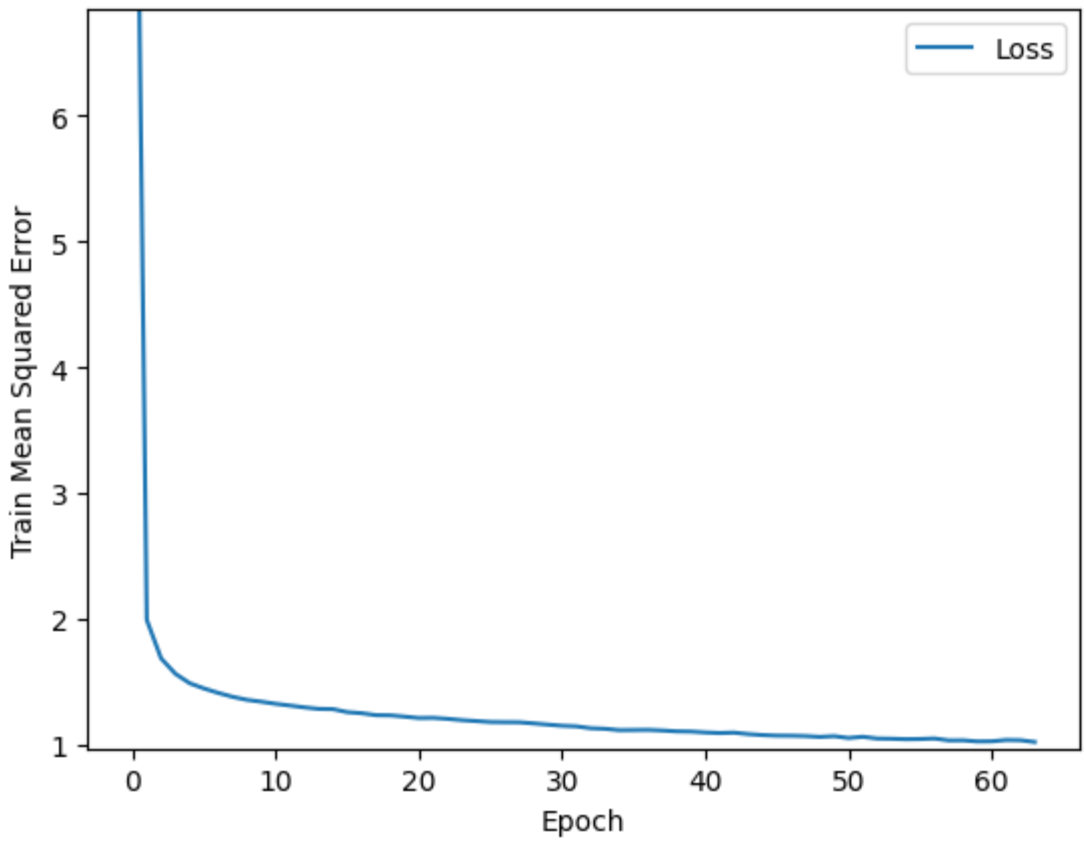
```
Epoch 1/64
127/127 [==============================] - 1s 663us/step - loss: 11.9089 - mean_squared_error: 11.4065 - m
ean_absolute_error: 2.4547
Epoch 2/64
127/127 [==============================] - 0s 600us/step - loss: 2.4395 - mean_squared_error: 1.9935 - mea
n_absolute_error: 0.9172
Epoch 3/64
127/127 [==============================] - 0s 606us/step - loss: 2.1079 - mean_squared_error: 1.6877 - mea
n_absolute_error: 0.7912
Epoch 4/64
127/127 [==============================] - 0s 605us/step - loss: 1.9676 - mean_squared_error: 1.5652 - mea
n_absolute_error: 0.7303
Epoch 5/64
127/127 [==============================] - 0s 613us/step - loss: 1.8802 - mean_squared_error: 1.4902 - mea
n_absolute_error: 0.6895
Epoch 6/64
127/127 [==============================] - 0s 599us/step - loss: 1.8266 - mean_squared_error: 1.4485 - mea
n_absolute_error: 0.6649
Epoch 7/64
127/127 [==============================] - 0s 597us/step - loss: 1.7789 - mean_squared_error: 1.4130 - mea
n_absolute_error: 0.6500
Epoch 8/64
127/127 [==============================] - 0s 614us/step - loss: 1.7378 - mean_squared_error: 1.3822 - mea
n_absolute_error: 0.6364
Epoch 9/64
127/127 [==============================] - 0s 642us/step - loss: 1.7032 - mean_squared_error: 1.3582 - mea
n_absolute_error: 0.6261
Epoch 10/64
127/127 [==============================] - 0s 614us/step - loss: 1.6784 - mean_squared_error: 1.3436 - mea
n_absolute_error: 0.6204
Epoch 11/64
127/127 [==============================] - 0s 616us/step - loss: 1.6511 - mean_squared_error: 1.3267 - mea
n_absolute_error: 0.6177
Epoch 12/64
127/127 [==============================] - 0s 619us/step - loss: 1.6290 - mean_squared_error: 1.3129 - mea
n_absolute_error: 0.6116
Epoch 13/64
127/127 [==============================] - 0s 639us/step - loss: 1.6070 - mean_squared_error: 1.2985 - mea
n_absolute_error: 0.6106
Epoch 14/64
127/127 [==============================] - 0s 626us/step - loss: 1.5880 - mean_squared_error: 1.2865 - mea
n_absolute_error: 0.6048
```

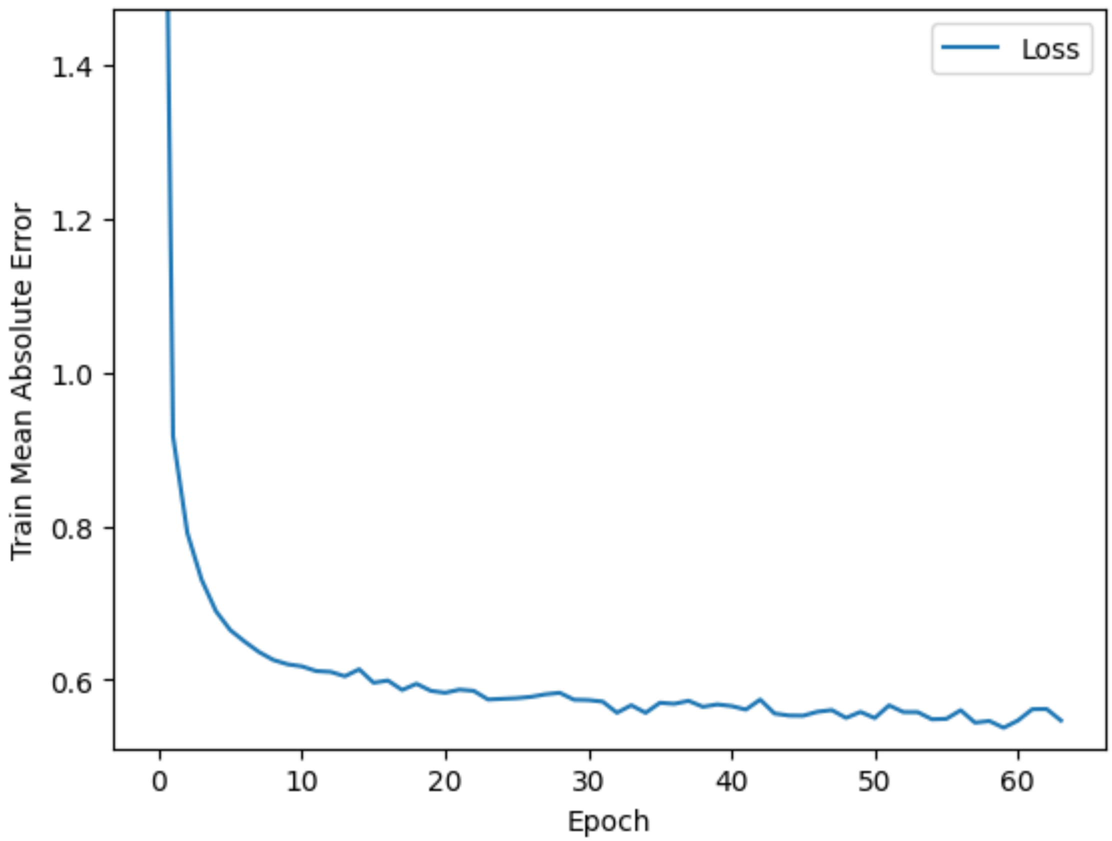
```
Epoch 15/64
127/127 [==============================] - 0s 611us/step - loss: 1.5791 - mean_squared_error: 1.2846 - mea
n_absolute_error: 0.6139
Epoch 16/64
127/127 [==============================] - 0s 609us/step - loss: 1.5487 - mean_squared_error: 1.2606 - mea
n_absolute_error: 0.5964
Epoch 17/64
127/127 [==============================] - 0s 611us/step - loss: 1.5350 - mean_squared_error: 1.2522 - mea
n_absolute_error: 0.5993
Epoch 18/64
127/127 [==============================] - 0s 611us/step - loss: 1.5134 - mean_squared_error: 1.2366 - mea
n_absolute_error: 0.5870
Epoch 19/64
127/127 [==============================] - 0s 617us/step - loss: 1.5069 - mean_squared_error: 1.2352 - mea
n_absolute_error: 0.5951
Epoch 20/64
127/127 [==============================] - 0s 613us/step - loss: 1.4920 - mean_squared_error: 1.2251 - mea
n_absolute_error: 0.5859
Epoch 21/64
127/127 [==============================] - 0s 623us/step - loss: 1.4769 - mean_squared_error: 1.2138 - mea
n_absolute_error: 0.5833
Epoch 22/64
127/127 [==============================] - 0s 605us/step - loss: 1.4736 - mean_squared_error: 1.2152 - mea
n_absolute_error: 0.5874
Epoch 23/64
127/127 [==============================] - 0s 613us/step - loss: 1.4622 - mean_squared_error: 1.2075 - mea
n_absolute_error: 0.5858
Epoch 24/64
127/127 [==============================] - 0s 618us/step - loss: 1.4477 - mean_squared_error: 1.1971 - mea
n_absolute_error: 0.5746
Epoch 25/64
127/127 [==============================] - 0s 646us/step - loss: 1.4373 - mean_squared_error: 1.1899 - mea
n_absolute_error: 0.5754
Epoch 26/64
127/127 [==============================] - 0s 621us/step - loss: 1.4271 - mean_squared_error: 1.1823 - mea
n_absolute_error: 0.5763
Epoch 27/64
127/127 [==============================] - 0s 607us/step - loss: 1.4216 - mean_squared_error: 1.1808 - mea
n_absolute_error: 0.5780
Epoch 28/64
127/127 [==============================] - 0s 605us/step - loss: 1.4183 - mean_squared_error: 1.1801 - mea
n_absolute_error: 0.5813
```

```
Epoch 29/64
127/127 [==============================] - 0s 596us/step - loss: 1.4065 - mean_squared_error: 1.1714 - mea
n_absolute_error: 0.5833
Epoch 30/64
127/127 [==============================] - 0s 602us/step - loss: 1.3939 - mean_squared_error: 1.1616 - mea
n_absolute_error: 0.5743
Epoch 31/64
127/127 [==============================] - 0s 595us/step - loss: 1.3834 - mean_squared_error: 1.1524 - mea
n_absolute_error: 0.5738
Epoch 32/64
127/127 [==============================] - 0s 622us/step - loss: 1.3763 - mean_squared_error: 1.1478 - mea
n_absolute_error: 0.5718
Epoch 33/64
127/127 [==============================] - 0s 596us/step - loss: 1.3597 - mean_squared_error: 1.1327 - mea
n_absolute_error: 0.5574
Epoch 34/64
127/127 [==============================] - 0s 606us/step - loss: 1.3539 - mean_squared_error: 1.1284 - mea
n_absolute_error: 0.5671
Epoch 35/64
127/127 [==============================] - 0s 628us/step - loss: 1.3419 - mean_squared_error: 1.1175 - mea
n_absolute_error: 0.5573
Epoch 36/64
127/127 [==============================] - 0s 609us/step - loss: 1.3418 - mean_squared_error: 1.1185 - mea
n_absolute_error: 0.5703
Epoch 37/64
127/127 [==============================] - 0s 628us/step - loss: 1.3400 - mean_squared_error: 1.1197 - mea
n_absolute_error: 0.5690
Epoch 38/64
127/127 [==============================] - 0s 601us/step - loss: 1.3340 - mean_squared_error: 1.1148 - mea
n_absolute_error: 0.5729
Epoch 39/64
127/127 [==============================] - 0s 600us/step - loss: 1.3254 - mean_squared_error: 1.1082 - mea
n_absolute_error: 0.5651
Epoch 40/64
127/127 [==============================] - 0s 603us/step - loss: 1.3224 - mean_squared_error: 1.1065 - mea
n_absolute_error: 0.5682
Epoch 41/64
127/127 [==============================] - 0s 608us/step - loss: 1.3138 - mean_squared_error: 1.0993 - mea
n_absolute_error: 0.5661
Epoch 42/64
127/127 [==============================] - 0s 611us/step - loss: 1.3078 - mean_squared_error: 1.0948 - mea
n_absolute_error: 0.5614
```

```
Epoch 43/64
127/127 [==============================] - 0s 604us/step - loss: 1.3103 - mean_squared_error: 1.0979 - mea
n_absolute_error: 0.5744
Epoch 44/64
127/127 [==============================] - 0s 604us/step - loss: 1.2965 - mean_squared_error: 1.0862 - mea
n_absolute_error: 0.5564
Epoch 45/64
127/127 [==============================] - 0s 602us/step - loss: 1.2874 - mean_squared_error: 1.0784 - mea
n_absolute_error: 0.5537
Epoch 46/64
127/127 [==============================] - 0s 606us/step - loss: 1.2812 - mean_squared_error: 1.0740 - mea
n_absolute_error: 0.5535
Epoch 47/64
127/127 [==============================] - 0s 637us/step - loss: 1.2797 - mean_squared_error: 1.0733 - mea
n_absolute_error: 0.5586
Epoch 48/64
127/127 [==============================] - 0s 607us/step - loss: 1.2751 - mean_squared_error: 1.0705 - mea
n_absolute_error: 0.5608
Epoch 49/64
127/127 [==============================] - 0s 613us/step - loss: 1.2683 - mean_squared_error: 1.0640 - mea
n_absolute_error: 0.5506
Epoch 50/64
127/127 [==============================] - 0s 601us/step - loss: 1.2723 - mean_squared_error: 1.0698 - mea
n_absolute_error: 0.5584
Epoch 51/64
127/127 [==============================] - 0s 602us/step - loss: 1.2566 - mean_squared_error: 1.0552 - mea
n_absolute_error: 0.5505
Epoch 52/64
127/127 [==============================] - 0s 611us/step - loss: 1.2653 - mean_squared_error: 1.0650 - mea
n_absolute_error: 0.5670
Epoch 53/64
127/127 [==============================] - 0s 624us/step - loss: 1.2504 - mean_squared_error: 1.0510 - mea
n_absolute_error: 0.5582
Epoch 54/64
127/127 [==============================] - 0s 604us/step - loss: 1.2472 - mean_squared_error: 1.0491 - mea
n_absolute_error: 0.5581
Epoch 55/64
127/127 [==============================] - 0s 616us/step - loss: 1.2425 - mean_squared_error: 1.0457 - mea
n_absolute_error: 0.5488
Epoch 56/64
127/127 [==============================] - 0s 614us/step - loss: 1.2427 - mean_squared_error: 1.0471 - mea
n_absolute_error: 0.5494
```

```
Epoch 57/64
127/127 [==============================] - 0s 605us/step - loss: 1.2463 - mean_squared_error: 1.0513 - mea
n_absolute_error: 0.5606
Epoch 58/64
127/127 [==============================] - 0s 599us/step - loss: 1.2296 - mean_squared_error: 1.0361 - mea
n_absolute_error: 0.5445
Epoch 59/64
127/127 [==============================] - 0s 674us/step - loss: 1.2304 - mean_squared_error: 1.0367 - mea
n_absolute_error: 0.5466
Epoch 60/64
127/127 [==============================] - 0s 603us/step - loss: 1.2208 - mean_squared_error: 1.0281 - mea
n_absolute_error: 0.5378
Epoch 61/64
127/127 [==============================] - 0s 597us/step - loss: 1.2206 - mean_squared_error: 1.0290 - mea
n_absolute_error: 0.5473
Epoch 62/64
127/127 [==============================] - 0s 606us/step - loss: 1.2301 - mean_squared_error: 1.0393 - mea
n_absolute_error: 0.5619
Epoch 63/64
127/127 [==============================] - 0s 602us/step - loss: 1.2276 - mean_squared_error: 1.0376 - mea
n_absolute_error: 0.5622
Epoch 64/64
127/127 [==============================] - 0s 619us/step - loss: 1.2119 - mean_squared_error: 1.0235 - mea
n_absolute_error: 0.5473
```


Loss
Train Mean Squared Error
Epoch
1
2
3
4
5
6
0
10
20
30
40
50
60




```
'Evaluating the model against the test_df'
32/32 [==============================] - 0s 570us/step - loss: 1.2483 - mean_squared_error: 1.0598 - mean_
absolute_error: 0.5547
'Predicting the nn_origin_df and comparing with the initial data'
50532/50532 [==============================] - 19s 381us/step
'predictions'
array([0.83517975, 0.83517975, 0.83517975, ..., 0.83517975, 0.83517975,
       0.83517975], dtype=float32)
'predictions.shape'
(6467980,)
'real_ratings'
array([0., 0., 0., ..., 0., 0., 0.])
'real_mean_squared_error=0.5568353119072986, real_mean_absolute_error=0.6097937019215851'
'results_df'
```

| | TRAIN_MSE | TRAIN_MAE | TEST_MSE | TEST_MAE | REAL_MSE | REAL_MAE |
|---|---|---|---|---|---|---|
| **CLUSTER_0** | 0.825997 | 0.484262 | 0.914926 | 0.525575 | 0.504794 | 0.586051 |
| **CLUSTER_1** | 0.505148 | 0.297598 | 0.513924 | 0.307466 | 0.323048 | 0.438537 |
| **CLUSTER_2** | 0.955786 | 0.505149 | 0.979159 | 0.500075 | 0.495864 | 0.552092 |
| **CLUSTER_3** | 0.783235 | 0.472670 | 0.749991 | 0.490155 | 0.484604 | 0.571959 |
| **CLUSTER_4** | 1.023517 | 0.547274 | 1.059759 | 0.554747 | 0.556835 | 0.609794 |