#1)

#libraries

import os

import pandas as pd

import requests

import zipfile

#url for data

dataset\_url="http://files.grouplens.org/datasets/movielens/ml-25m.zip"

checksum\_url = 'http://files.grouplens.org/datasets/movielens/ml-25m.zip.md5'

#function to dowload files from url

def download\_file(url , filename):

r=requests.get(url)

with open(filename , "wb") as f:

f.write(r.content)

#function to unzpi the downloaded folder containing the datasets

def unzip\_file(zip\_file, extract\_dir):

with zipfile.ZipFile(zip\_file, 'r') as zip\_ref:

zip\_ref.extractall(extract\_dir)

#perfrom downloading and unzipping

download\_file(dataset\_url , "ml-25m.zip")

unzip\_file('ml-25m.zip', 'ml-25m')

download\_file(checksum\_url , "ml-25m.zip.md5")

with open('ml-25m.zip.md5', 'r') as file:

md5\_checksum = file.read().strip()

md5\_checksum=md5\_checksum.split()

md5\_checksum=md5\_checksum[0]

md5\_checksum

print([f for f in os.listdir("ml-25m/ml-25m")])

#2): If you had to design a relational database schema (SQL) to store your results, what would it be?

#To ansewer, first look at the data

df\_path="ml-25m/ml-25m/"

skip\_list="README.txt"

df\_list=[pd.read\_csv(df\_path+f) for f in os.listdir("ml-25m/ml-25m") if f not in skip\_list]

df\_list

links=df\_list[0]

tags=df\_list[1]

genome\_tags=df\_list[2]

ratings=df\_list[3]

genome\_scores=df\_list[4]

movies=df\_list[5]

#So, we need 6 tables to store data. Now we can look at additional information to check

print("Dataset name: Links\n")

print(f"Data types:\n\n{links.dtypes}\n")

print(f"Number of rows: {len(links)}\n\n")

print(f"Number of columns: {links.shape[1]}\n\n")

print(f"Unique entries per columns: {links.nunique()}")

print("Dataset name: Tags\n")

print(f"Data types:\n\n{tags.dtypes}\n")

print(f"Number of rows: {len(tags)}\n\n")

print(f"Number of columns: {tags.shape[1]}\n\n")

print(f"Unique entries per columns: {tags.nunique()}")

print("Dataset name: Genome\_Tags\n")

print(f"Data types:\n\n{genome\_tags.dtypes}\n")

print(f"Number of rows: {len(genome\_tags)}\n\n")

print(f"Number of columns: {genome\_tags.shape[1]}\n\n")

print(f"Unique entries per columns: {genome\_tags.nunique()}")

print("Dataset name: Genome\_Scores\n")

print(f"Data types:\n\n{genome\_scores.dtypes}\n")

print(f"Number of rows: {len(genome\_scores)}\n\n")

print(f"Number of columns: {genome\_scores.shape[1]}\n\n")

print(f"Unique entries per columns: {genome\_scores.nunique()}")

print("Dataset name: Ratings\n")

print(f"Data types:\n\n{ratings.dtypes}\n")

print(f"Number of rows: {len(ratings)}\n\n")

print(f"Number of columns: {ratings.shape[1]}\n\n")

print(f"Unique entries per columns: {ratings.nunique()}")

print("Dataset name: Movies\n")

print(f"Data types:\n\n{movies.dtypes}\n")

print(f"Number of rows: {len(movies)}\n\n")

print(f"Number of columns: {movies.shape[1]}\n\n")

print(f"Unique entries per columns: {movies.nunique()}")

#So i would design a relational databes with 6 tables, for which colums have the same data types

#as the data types they have in the dfs (so , for example, movieId attribute should be set to

#integer, title as a varchar, and so on).

#Additionally, we can set the primary keys for each table:

#The tables Movies and Links should have the attribute movieId as primary key, since it has

#a name of unique elements equal to the number of entries in the tables to it uniquely identify

#each row and we can't have a key made of less than one attribute. For the same reason, Genome\_Tags

#table should have tag\_id as primary key

#For the remaining columns, i would suggest to set as key a comnbination of different attributes.

#For example:

print(tags.columns)

tags.movieId=tags.movieId.astype(str)

tags.userId=tags.userId.astype(str)

tags.timestamp=tags.timestamp.astype(str)

tags.tag=tags.tag.astype(str)

tags["Candidate\_key"]=tags.movieId+"\_"+tags.timestamp+"\_"+tags.tag

tags.Candidate\_key.nunique()==len(tags)

#The combination of movieId, timestamp and tag makes a key for Tags table