In [2]:

```
import numpy as np
import pandas as pd
```

Reading dataset: We expect dataset to be present at datasets folder. For getting the dataset from kaggle, we execute the data_fetchong.ipynb file. It extracts and stores the CSVs in datasets directory

In [3]:

```
users_df = pd.read_csv("datasets/users-details-2023.csv")
users_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731290 entries, 0 to 731289
Data columns (total 16 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Mal ID            731290 non-null  int64  
 1   Username          731289 non-null  object 
 2   Gender            224383 non-null  object 
 3   Birthday          168068 non-null  object 
 4   Location          152805 non-null  object 
 5   Joined            731290 non-null  object 
 6   Days Watched      731282 non-null  float64
 7   Mean Score        731282 non-null  float64
 8   Watching          731282 non-null  float64
 9   Completed         731282 non-null  float64
 10  On Hold           731282 non-null  float64
 11  Dropped           731282 non-null  float64
 12  Plan to Watch     731282 non-null  float64
 13  Total Entries     731282 non-null  float64
 14  Rewatched         731282 non-null  float64
 15  Episodes Watched  731282 non-null  float64
dtypes: float64(10), int64(1), object(5)
memory usage: 89.3+ MB
```

In [4]:

```
anime_df = pd.read_csv('datasets/anime-dataset-2023.csv')
anime_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24905 entries, 0 to 24904
Data columns (total 24 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   anime_id      24905 non-null  int64 
 1   Name          24905 non-null  object
 2   English name  24905 non-null  object
 3   Other name    24905 non-null  object
 4   Score         24905 non-null  object
 5   Genres        24905 non-null  object
 6   Synopsis      24905 non-null  object
 7   Type          24905 non-null  object
 8   Episodes      24905 non-null  object
 9   Aired         24905 non-null  object
 10  Premiered     24905 non-null  object
 11  Status        24905 non-null  object
 12  Producers     24905 non-null  object
 13  Licensors     24905 non-null  object
 14  Studios       24905 non-null  object
 15  Source        24905 non-null  object
 16  Duration      24905 non-null  object
 17  Rating        24905 non-null  object
 18  Rank          24905 non-null  object
 19  Popularity    24905 non-null  int64 
 20  Favorites     24905 non-null  int64 
 21  Scored By     24905 non-null  object
 22  Members       24905 non-null  int64 
 23  Image URL     24905 non-null  object
dtypes: int64(4), object(20)
memory usage: 4.6+ MB
```

In [5]:

```
user_score_df = pd.read_csv('datasets/users-score-2023.csv')
user_score_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24325191 entries, 0 to 24325190
Data columns (total 5 columns):
 #   Column       Dtype 
---  ------       ----- 
 0   user_id      int64 
 1   Username     object
 2   anime_id     int64 
 3   Anime Title  object
 4   rating       int64 
dtypes: int64(3), object(2)
memory usage: 927.9+ MB
```

Cleaning Users Details dataset for Joining Date and Birthday

1. Cleaning joining date. Steps followed for these are:
   - Remove rows where the value is NaN
   - Remove time strings from date strings
   - Convert Date strings to pandas datetime object
2. Cleaning birthdate. Steps followed for these are:

- Remove rows where the value is NaN
- Remove time strings from date strings
- Convert Date strings to pandas datetime object
- Remove entries where birthdate is very very old, ie before 1950-01-01

3. Cleaning entries based on relation between joining date and birthdate
    - Remove entries where joining date is older than birthdate
    - Remove entries where age at joining is less than 5 years.

```
In [6]: df_users_cleaned = users_df[~users_df["Birthday"].isna()]
        df_users_cleaned = df_users_cleaned[~df_users_cleaned["Joined"].isna()]

        # Remove time stamp from date string
        df_users_cleaned["Birthday_Date"] = df_users_cleaned["Birthday"].str.slice(0
        df_users_cleaned["Joined_Date"] = df_users_cleaned["Joined"].str.slice(0,10)

        # Convert string to date time object
        df_users_cleaned["Birthday_Date"] = pd.to_datetime(df_users_cleaned["Birthda
        df_users_cleaned["Joined_Date"] = pd.to_datetime(df_users_cleaned["Joined_Da

        # Remove entries where joining date is before birthdate
        df_users_cleaned = df_users_cleaned[df_users_cleaned["Birthday_Date"] < df_u

        # Remove entries where birthday is very very old
        df_users_cleaned = df_users_cleaned[df_users_cleaned["Birthday_Date"] > pd.t

        # Remove entries where age at joining is less than 5 years
        df_users_cleaned["Age_Join"] = (df_users_cleaned["Joined_Date"] - df_users_c
        df_users_cleaned = df_users_cleaned[df_users_cleaned["Age_Join"]>5]

        df_users_cleaned[["Birthday_Date","Joined_Date", "Age_Join"]].sort_values(by
```

Out[6]:

	Birthday_Date	Joined_Date	Age_Join
644052	2006-12-24	2011-12-22	5.010989
689583	2007-03-07	2012-03-06	5.016484
298281	2005-07-16	2010-07-16	5.016484
164355	2004-04-12	2009-04-29	5.063187
22741	2002-10-07	2007-11-29	5.162088
...	...	...	...
652659	1950-08-22	2012-01-03	61.576923
444033	1950-02-20	2011-07-05	61.579670
717227	1950-11-14	2012-04-04	61.598901
668048	1950-03-23	2012-01-25	62.054945
715537	1950-04-25	2012-04-02	62.151099

166200 rows × 3 columns

Cleaning Anime Dataset

The **Aired** attribute is very important for us. Our target is to extract information like how long an anime runs, is it still ongoing, how many episodes does it have. The aim is to extract start date and end date of an anime and add those 2 as new columns to the dataframe: - Split the date string using the word **to** and strip white spaces - convert both Start date and end date to datetime objects - Inserted the new columns to the original dataframe

In [7]:

```python
# split the date string using the word **to**
aired = anime_df['Aired'].str.split('to', expand=True)
# Then strip whitespaces
aired[0] = aired[0].str.strip()
aired[1] = aired[1].str.strip()

# Finally convert both Start date and end date to datetime objects
aired[0] = pd.to_datetime(aired[0], format='%b %d, %Y', errors='coerce')
aired[1] = pd.to_datetime(aired[1], format='%b %d, %Y', errors='coerce')

# Rename the clomns
aired.rename(columns={0: 'Start Date', 1: 'End Date'}, inplace=True)

# Inserted the new columns to the original dataframe
anime_df.insert(10, 'Start Date', aired['Start Date'])
anime_df.insert(11, 'End Date', aired['End Date'])

anime_df[["Aired", "Start Date", "End Date"]].head()
```

Out[7]:

	Aired	Start Date	End Date
0	Apr 3, 1998 to Apr 24, 1999	1998-04-03	1999-04-24
1	Sep 1, 2001	2001-09-01	NaT
2	Apr 1, 1998 to Sep 30, 1998	1998-04-01	1998-09-30
3	Jul 3, 2002 to Dec 25, 2002	2002-07-03	2002-12-25
4	Sep 30, 2004 to Sep 29, 2005	2004-09-30	2005-09-29

The aim is to add a new cloumn named **Ongoing**. The way we do this is the aired column has format from start date to end date. The end date has ? for ongoing animes. Hence the rows having ? are tagged as ongoing animes This helps us in: - Knowing if an anime is still ongoing - Calculating number of episodes in an anime, for ongoing animes the dataset does not have number of episodes.

In [8]:

```
def check(value):
    return 1 if '?' in value else 0

anime_df['Ongoing'] = anime_df['Aired'].apply(check)
anime_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24905 entries, 0 to 24904
Data columns (total 27 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   anime_id      24905 non-null  int64         
 1   Name          24905 non-null  object        
 2   English name  24905 non-null  object        
 3   Other name    24905 non-null  object        
 4   Score         24905 non-null  object        
 5   Genres        24905 non-null  object        
 6   Synopsis      24905 non-null  object        
 7   Type          24905 non-null  object        
 8   Episodes      24905 non-null  object        
 9   Aired         24905 non-null  object        
 10  Start Date    20090 non-null  datetime64[ns]
 11  End Date      9337 non-null   datetime64[ns]
 12  Premiered     24905 non-null  object        
 13  Status        24905 non-null  object        
 14  Producers     24905 non-null  object        
 15  Licensors     24905 non-null  object        
 16  Studios       24905 non-null  object        
 17  Source        24905 non-null  object        
 18  Duration      24905 non-null  object        
 19  Rating        24905 non-null  object        
 20  Rank          24905 non-null  object        
 21  Popularity    24905 non-null  int64         
 22  Favorites     24905 non-null  int64         
 23  Scored By     24905 non-null  object        
 24  Members       24905 non-null  int64         
 25  Image URL     24905 non-null  object        
 26  Ongoing       24905 non-null  int64         
dtypes: datetime64[ns](2), int64(5), object(20)
memory usage: 5.1+ MB
```

The episodes field is also very important for us. We can infer whether people like short animes or long animes based on number of episodes. However some records of our dataset have "UNKNOWN" in the episodes field, this is because the anime is currently running. Just for analysis purpose, we get the episode count till jan 01 2024, since each episode is released once in a week we divide the aired duration by 1 week

In [9]:

```
for index, row in anime_df.iterrows():
    if row['Episodes'] == 'UNKNOWN':
        anime_df.loc[index, 'Episodes'] = ((pd.to_datetime('Jan 01, 2024', f

anime_df[anime_df['Episodes'] == 'UNKNOWN']
```

Out[9]:

anime_id	Name	English name	Other name	Score	Genres	Synopsis	Type	Episodes	Aired

0 rows × 27 columns

we normalize episodes field so that we can bring it to a common scale for comparing between different animes

We use MinMax normalization which shrinks the scale between 0 to 1. X_norm = (X−min(X))/(max(X)−min(X))

In [10]:
```
anime_df['Episodes'] = anime_df['Episodes'].astype(float)
anime_df['Episodes'] = (anime_df['Episodes'] - anime_df['Episodes'].min()) /
anime_df['Episodes']
```

Out[10]:
```
0        0.008181
1        0.000000
2        0.008181
3        0.008181
4        0.016688
           ...
24900    0.004581
24901    0.005563
24902    0.004908
24903    0.000000
24904    0.000000
Name: Episodes, Length: 24905, dtype: float64
```

Cleaning User Details for Location of User

The Location data is very unstructured, some have country name, some have country suffix, some have name of state and so on. We try to fetch the country name from this in multiple ways.

1. We get the list of current countries and country codes using an API and then match the value in location column if it contains the code or country name.
2. For US states, we map all us states to USA country and then check the location field for these states.

In [13]:
```
import requests
import collections

url = 'https://restcountries.com/v3.1/all'
response = requests.get(url)
countries = collections.defaultdict(str)

if response.status_code == 200:
    response_body = response.json()

    for i in range(len(response_body)):
        common = response_body[i]['name']['common']
        official = response_body[i]['name']['official']

        countries[common] = common

        if 'nativeNames' in response_body[i]:
            native_names = response_body[i]['name']['nativeName']
            for key, val in native_names.items():
                countries[val['common']] = common
                countries[val['official']] = common

        if 'translations' in response_body[i]:
            translations = response_body[i]['translations']

            for key, val in translations.items():
                countries[val['common']] = common
                countries[val['official']] = common
else:
    print("Fail")

us_states = [
    'Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado', 'C
    'Hawaii', 'Idaho', 'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 
    'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Mo
    'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohio', 'Okl
    'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Wa
]

for state in us_states:
    countries[state] = 'United States'
```

In [14]:

```
for index, row in users_df.iterrows():

    location = row['Location']

    # To replace row whose type is not string with an empty string
    if type(location) != str:
        location = ""

    location_splited = location.split(",")
    is_valid_country = False

    for splited_str in location_splited:
        splited_str = splited_str.strip()

        if splited_str in countries:
            users_df.at[index, 'Location'] = countries[splited_str]
            is_valid_country = True

    if not is_valid_country:
        users_df.at[index, 'Location'] = None

print(users_df['Location'].notna().sum())
```

```
82759
```

Clean User Details to trim number of users

We trim the rows from user details to contain details of only the users for whom we have the user score data. Because without user score we cannot link a user data to anime data.

In [15]:

```
common_users = pd.merge(user_score_df, users_df, left_on=['user_id', 'Userna


users_df = users_df[users_df.set_index(['Mal ID', 'Username']).index.isin(co

print(users_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 264069 entries, 0 to 731289
Data columns (total 16 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   Mal ID           264069 non-null  int64  
 1   Username         264068 non-null  object 
 2   Gender           140554 non-null  object 
 3   Birthday         103198 non-null  object 
 4   Location         53217 non-null   object 
 5   Joined           264069 non-null  object 
 6   Days Watched     264067 non-null  float64
 7   Mean Score       264067 non-null  float64
 8   Watching         264067 non-null  float64
 9   Completed        264067 non-null  float64
 10  On Hold          264067 non-null  float64
 11  Dropped          264067 non-null  float64
 12  Plan to Watch    264067 non-null  float64
 13  Total Entries    264067 non-null  float64
 14  Rewatched        264067 non-null  float64
 15  Episodes Watched 264067 non-null  float64
dtypes: float64(10), int64(1), object(5)
memory usage: 34.2+ MB
None
```

Clean Genres: In anime dataset we have a few anime where the genre is "UNKNOWN". Currently we will exclude animes whose genre is "UNKNOWN" but later we want to look into getting genre for such anime using its synopsis.

In [20]:

```
anime_df = anime_df[~anime_df["Genres"].isna()]

anime_df = anime_df[anime_df['Genres'] != 'UNKNOWN']
anime_df[anime_df['Genres'] == 'UNKNOWN']
```

Out[20]:

anime_id	Name	English name	Other name	Score	Genres	Synopsis	Type	Episodes	Aired

0 rows × 27 columns

Storing the cleaned datasets in a new directory called cleaned_datasets

In [21]:

```
!mkdir cleaned_datasets
anime_df.to_csv("cleaned_datasets/anime_dataset_cleaned.csv")
users_df.to_csv("cleaned_datasets/users_details_dataset_cleaned.csv")
user_score_df.to_csv("cleaned_datasets/user_scores_cleaned.csv")
```

In [ ]:

```
```