Step 1A

- This dataset contains user preference data from 73516 users on 12294 anime.
- anime_id - myanimelist.net's unique id identifying an anime.
- name - full name of anime.
- genre - comma separated list of genres for this anime.
- type - movie, TV, OVA, etc.
- episodes - how many episodes in this show. (1 if movie).
- rating - average rating out of 10 for this anime.
- members - number of community members that are in this anime's "group".

Step 1B

We are interested in creating a recommender system based on this dataset. We might create this by:

- Given a genre, return a random anime that satisfies that genre
- Given a genre and minimum user rating, return a random anime that satisfies both inputs
- Given a genre, minimum user rating, and type, return a random anime that satisfies all three inputs

The idea I will choose to build on is:

- Given a genre, minimum user rating, and type, return a random anime that satisfies all three inputs

Step 1C

An example of the final function would be:

expect(give_anime("anime.csv", "Action", 8.65, "TV"), 'Cowboy Bebop')

Step 2A

- The columns 'name', 'genre', 'rating', and 'type' will be represented in my data definitions as the anime recommender will require a genre, minimum rating, and type as an input and the name of a random anime as an output.

In [1]:
```
from cs173 import *
from typing import NamedTuple, List
import csv

###################
# Data Definitions

Anime = NamedTuple('Anime', [('name', str),
                             ('genre', str),
                             ('rating', float), # in range [0.00 to 10.0]
                             ('type', str)])

# interp. an anime with it's name, a comma separated list of it's genres, rating, and the
# type of the anime

A1 = Anime("Death Note", "Mystery, Police, Psychological, Supernatural, Thriller", 8.71, "TV")
A2 = Anime("Kuroko no Basket", "Comedy, School, Shounen, Sports", 8.46, "TV")
A3 = Anime("Dragon Ball", "Adventure, Comedy, Fantasy, Martial Arts, Shounen, Super Power", 8.16, "TV")

# Template based on Compound (4 fields)
@typecheck
def fn_for_anime(a: Anime) -> ...:
    return ...(a.name,
               a.genre,
               a.rating,
               a.type)

# List[Anime]
# interp. a list of anime

LOA1 = []
LOA2 = [A1, A2, A3]
LOA_SMALL1 = [Anime('Kizna‌iver', 'Drama, Sci-Fi', 7.67, 'TV'),
              Anime('Gintama', 'Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen', 9.04, 'TV')]
LOA_SMALL2 = [Anime('Kizna‌iver', 'Drama, Sci-Fi', 7.67, 'TV'),
              Anime('Gintama', 'Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen', 9.04, 'TV'),
              Anime('Kiki no Na wa.', 'Drama, Romance, School, Supernatural', 9.37, 'Movie'),
              Anime('Cowboy Bebop', 'Action, Adventure, Comedy, Drama, Sci-Fi, Space', 8.82, 'TV'),
              Anime('One Punch Man', 'Action, Comedy, Parody, Sci-Fi, Seinen, Super Power, Supernatural', 8.82, 'TV')]

# Template based on arbitrary-sized and reference rule
@typecheck
def fn_for_loa(loa: List[Anime]) -> ...:
    # description of the accumulator
    acc = ... # type: ...

    for a in loa:
        acc = ...(fn_for_anime(a), acc)

    return ...(acc)
```

```
<>:23: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
<>:48: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
<>:50: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
<>:47: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
<>:48: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
<>:50: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
<ipython-input-1-1185173651?b-23: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
  return ...(a.name,
<ipython-input-1-1185173651?b-48: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
  acc = ...(fn_for_anime(a), acc)
<ipython-input-1-1185173651?b-50: SyntaxWarning: 'ellipsis' object is not callable; perhaps you missed a comma?
  return ...(acc)
```

Step 2B

In [2]:
```
@typecheck
def read(filename: str) -> List[Anime]:
    """
    reads information from a specified file and returns a list of anime,
    with each anime having a name, list of genres as a string, a float
    representing it's user rating score out of 10, and the anime's type
    """
    # return [] # stub

    # Template from HtDAP

    # loa contains the anime seen so far
    loa = []   # type: List[Anime]

    with open(filename, encoding = 'utf-8') as csvfile:

        reader = csv.reader(csvfile)
        next(reader) # skip header line

        for row in reader:
            a = Anime(row[1], row[2], (parse_float(row[5])), (row[3]))
            loa.append(a)

    return loa

start_testing()

# Examples and tests for read
expect(read("test_empty.csv"), [])
expect(read("anime_small.csv"), LOA_SMALL1)
expect(read("anime_small2.csv"), LOA_SMALL2)

summary()
```

```
3 of 3 tests passed
```

Step 2C

In [3]:
```
import random
COMMON_GENRES = ['Action', 'Adventure', 'Comedy', 'Shounen', 'Romance', 'Fantasy', 'Drama']

@typecheck
def give_anime(filename: str, genre: str, min_rating: float, anime_type: str) -> str:
    """
    Reads the file from the given filename, analyzes the data,
    and returns a random anime based on the given genre and
    the given minimum rating
    """
    # return "" # stub
    # Template based on HtDAP, based on function composition

    return analyze(read(filename), genre, min_rating, anime_type)

@typecheck
def analyze(loa: List[Anime], genre: str, min_rating: float, anime_type: str) -> str:
    """
    Return the name of a random anime from a given genre
    and minimum user rating from loa
    """
    # return "" # stub
    # Template based on composition

    filtered_list_genre = filter_anime_by_genre(loa, genre)
    filtered_list_rating = filter_anime_by_rating(filtered_list_genre, min_rating)
    filtered_list_type = filter_anime_by_type(filtered_list_rating, anime_type)
    filtered_list_name = only_anime_names(filtered_list_type)
    if filtered_list_name == []:
        return "Sorry! There are no anime with these specifications! Try again!"
    else:
        return random.choice(filtered_list_name)

@typecheck
def filter_anime_by_genre(loa: List[Anime], genre: str) -> List[Anime]:
    """
    Filter loa by the given genre
    """
    # return [] # stub
    # Template based on List[Anime] with an additional parameter (genre)
    # anime_contains a list of the anime seen so far
    anime = [] # type: List[Anime]

    for a in loa:
        if genre_matches(a, genre):
            anime.append(a)

    return anime

@typecheck
def genre_matches(a: Anime, genre: str) -> bool:
    """
    Return True if genre is in a.genre, False otherwise
    """
    # return True # stub
    # Template based on Anime with an additional parameter (genre)
    return genre in a.genre

@typecheck
def filter_anime_by_rating(loa: List[Anime], min_rating: float) -> List[Anime]:
    """
    Return a list of anime above the given minimum user rating
    """
    # return [] # stub
    # Template based on List[Anime] with an additional parameter (min_rating)
    # anime_above_min_rating stores the anime above the minimum rating seen so far
    anime_above_min_rating = [] # type: List[Anime]

    for a in loa:
        if above_rating(a, min_rating):
            anime_above_min_rating.append(a)

    return anime_above_min_rating

@typecheck
def above_rating(a: Anime, min_rating: float) -> bool:
    """
    Return True if a.rating is greater than min_rating
    If the anime has no rating, it will be given a score
    of 0.
    """
    # return False # stub
    # Template based on Anime with an additional parameter (min_rating)
    anime_no_rate = 0
    if a.rating == None:
        return anime_no_rate > min_rating
    elif min_rating == None:
        min_rating = 0
        return a.rating > anime_no_rate
    else:
        return a.rating > min_rating

@typecheck
def filter_anime_by_type(loa: List[Anime], anime_type: str) -> List[Anime]:
    """
    Filter loa by the given type of the anime (e.g. TV, Movie, etc.)
    """
    # return [] # stub
    # Template based on List[Anime] with an additional parameter (anime_type)
    # anime_given_type stores the anime of the given anime_type seen so far
    anime_given_type = [] # type: List[Anime]

    for a in loa:
        if anime_type_matches(a, anime_type):
            anime_given_type.append(a)

    return anime_given_type

@typecheck
def anime_type_matches(a: Anime, anime_type: str) -> bool:
    """
    Return True if a.type == anime_type, False otherwise
    """
    # return True # stub
    # Template based on Anime with an additional parameter (anime_type)
    if a.type == None:
        return False
    else:
        return a.type == anime_type

@typecheck
def only_anime_names(loa: List[Anime]) -> List[str]:
    """
    Return a list of anime names from loa
    """
    # return [] # stub
    # Template based on List[Anime]
    # anime_names contains the anime names seen so far
    anime_names = [] # type: List[str]

    for a in loa:
        anime_names.append(a.name)

    return anime_names



start_testing()

# Examples and tests for filter_anime_by_genre
expect(filter_anime_by_genre(LOA_SMALL1, "Action"),
       [Anime('Gintama', 'Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen', 9.04, 'TV')])
expect(filter_anime_by_genre(LOA_SMALL2, "Comedy"),
       [Anime('Gintama', 'Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen', 9.04, 'TV'),
        Anime('Cowboy Bebop', 'Action, Adventure, Comedy, Drama, Sci-Fi, Space', 8.82, 'TV'),
        Anime('One Punch Man', 'Action, Comedy, Parody, Sci-Fi, Seinen, Super Power, Supernatural', 8.82, 'TV')])
expect(filter_anime_by_genre([], "Supernatural"), [])

summary()

start_testing()

# Examples and tests for genre_matches
expect(genre_matches(A1, "Thriller"), True)
expect(genre_matches(A2, "Fantasy"), False)
expect(genre_matches(A3, "Martial Arts"), True)

summary()

start_testing()

# Examples and tests for filter_anime_by_rating
expect(filter_anime_by_rating([], 4.21), [])
expect(filter_anime_by_rating(LOA_SMALL1, 8.05),
       [Anime('Gintama', 'Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen', 9.04, 'TV')])
expect(filter_anime_by_rating(LOA_SMALL2, 8.82),
       [Anime('Gintama', 'Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen', 9.04, 'TV'),
        Anime('Kiki no Na wa.', 'Drama, Romance, School, Supernatural', 9.37, 'Movie')])

summary()

start_testing()

# Examples and tests for above_rating
expect(above_rating(A1, 3.64), True)
expect(above_rating(A2, 9.32), False)
expect(above_rating(A3, 8.16), False)

summary()

start_testing()

# Examples and tests for filter_anime_by_type
expect(filter_anime_by_type(LOA_SMALL2, 'TV'),
       [Anime('Kizna‌iver', 'Drama, Sci-Fi', 7.67, 'TV'),
        Anime('Gintama', 'Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen', 9.04, 'TV'),
        Anime('Cowboy Bebop', 'Action, Adventure, Comedy, Drama, Sci-Fi, Space', 8.82, 'TV'),
        Anime('One Punch Man', 'Action, Comedy, Parody, Sci-Fi, Seinen, Super Power, Supernatural', 8.82, 'TV')])

expect(filter_anime_by_type(LOA_SMALL2, 'Movie'), [Anime('Kiki no Na wa.', 'Drama, Romance, School, Supernatural', 9.37, 'Movie')])

summary()

start_testing()

# Examples and tests for anime_type_matches
expect(anime_type_matches(A1, 'Movie'), False)
expect(anime_type_matches(A2, 'TV'), True)

summary()

start_testing()

# Examples and tests for only_anime_names
expect(only_anime_names([]), [])
expect(only_anime_names(LOA_SMALL1), ['Kizna‌iver', 'Gintama'])
expect(only_anime_names(LOA_SMALL2), ['Kizna‌iver', 'Gintama', 'Kiki no Na wa.', 'Cowboy Bebop', 'One Punch Man'])

summary()
```

```
3 of 3 tests passed
3 of 3 tests passed
3 of 3 tests passed
3 of 3 tests passed
2 of 2 tests passed
3 of 3 tests passed
3 of 3 tests passed
```

In [4]:
```
give_anime("anime.csv", "Action", 8.65, "TV")
```

Out[4]:
```
'Cowboy Bebop'
```

In [12]:
```
x = input("What genre of anime do you want to watch? ")
y = input("What should the minimum rating of the anime be? (out of 10) ")
z = input("Do you want to watch a TV show or a movie? Enter 'TV' for a TV show, or 'Movie' for a movie: ")
result = give_anime("anime.csv", x, float(y), z)
print(result)
```

```
What genre of anime do you want to watch? Action
What should the minimum rating of the anime be? (out of 10) 8
Do you want to watch a TV show or a movie? Enter 'TV' for a TV show, or 'Movie' for a movie: TV
Fullmetal Alchemist: Brotherhood
```

In [ ]:
```
```