# Data Analytics Project

Code

*19 - Musical Instruments Recommendation*

*Alessia Ruggeri*

# Introduction

For my Data Analytics Project, my task is to build at least two recommendation systems using a Musical Instruments dataset and to use them to predict the product ratings for some users based on other users' ratings. Two datasets have been given to me: the first one contains information about user preferences on musical instruments (ratings); the second one contains information about each rated item (metadata).

My main tasks are:

- Explore and describe the data through some standard descriptive statistics and some graphs;
- Pre-process the data;
- Build at least two recommendation systems for musical instruments ratings;
- Split the data into training and test sets and predict the variable 'rating' in the test set;
- Evaluate and compare the accuracy of the different models.

# Overview

A recommendation system is an information filtering system that aims to predict user responses to option; in particular, it tries to predict the user preference or rating of an item. It can be used to offer users custom suggestions about which items they might like, based on similarity between other users or other items, or based on a custom "profile" of the user or of the item.

The recommendation systems can be classified in two main groups:

- Content-based systems: they base their suggestions on the properties of the items recommended;
- Collaborative filtering systems: they recommend items based on similarity between users or between items.

# Exploration and description of data

First, I import all the packages that I need in order to handle the dataset.

Code

Then, I start by importing the datasets; to do this, I convert the .csv file containing the ratings and the .json file containing the metadata into two dataframes.

Hide

```
ds = read.csv(file = "ratings.csv")
colnames(ds) = c("user", "item", "rating", "timestamp")
ds['timestamp'] = as.POSIXct(ds[,'timestamp'], origin="1970-01-01")
```

Hide

```
json = readLines("new_meta_Musical_Instruments.json")
metadata = fromJSON(json)
```

In order to have an idea of the type of data I'm going to work with, I visualize the head and a summary of data.

Code

| | user<br><fctr> | item<br><fctr> | rating<br><dbl> | timestamp<br><S3: POSIXct> |
|---|---|---|---|---|
| 1 | A3TS466QBAWB9D | 0014072149 | 5 | 2013-06-06 02:00:00 |
| 2 | A3BUDYITWUSIS7 | 0041291905 | 5 | 2013-10-14 02:00:00 |
| 3 | A19K10Z0D2NTZK | 0041913574 | 5 | 2010-09-23 02:00:00 |
| 4 | A14X336IB4JD89 | 0201891859 | 1 | 2012-10-17 02:00:00 |
| 5 | A2HR0IL3TC4CKL | 0577088726 | 5 | 2013-06-14 02:00:00 |
| 6 | A2DHYD72O52WS5 | 0634029231 | 3 | 2005-06-24 02:00:00 |

6 rows

Code

```
            user                item            rating        timestamp

 A2PAD826IH1HFE:   483   B000ULAP4U:  3523   Min.   :1.000   Min.   :1998-04-25 02:0
0:00
 A2AIMXT9PLAM12:   463   B003VWJ2K8:  2275   1st Qu.:4.000   1st Qu.:2011-12-28 01:0
0:00
 A2NYK9KWFMJV4Y:   454   B003VWKPHC:  1603   Median :5.000   Median :2013-03-27 01:0
0:00
 A33GGROUQRQZS :   154   B001MSS6CS:  1420   Mean   :4.244   Mean   :2012-08-10 03:0
5:21
 A2PR6NXG0PA3KY:   135   B00FPPQYXM:  1287   3rd Qu.:5.000   3rd Qu.:2013-12-28 01:0
0:00
 ALAJL3S09HBS7 :   126   1417030321:  1218   Max.   :5.000   Max.   :2014-07-23 02:0
0:00
 (Other)       :498360   (Other)   :488849
```

Code

```
Rating standard deviation: 1.203374
```

Hide

```
# head(metadata)                # I leave this two lines commented because
# summary(metadata)             the output is too big for the report
```

The `csv` dataset contains 4 attributes, which are *user*, *item*, *rating* and *timestamp*, and 500,175 observations, each one representing an item's rating gaved by a user. This is the most interesting dataset, since it relates users and items by the corresponding rating, therefore I will work mostly on this data.

The `json` dataset contains 9 attributes, which are *asin* (item id), *title*, *price*, *imUrl*, *salesRank*, *categories*, *description*, *related* and *brand*, and 84,901 observations, each one representing the metadata of one item. Some attributes in turn contain a list of elements, like *salesRank* and *categories*. After I deeply explored this

dataset, I realised that it doesn't contain very interesting data for my purpose, furthermore more than a half of the metadata values are NA and can't be used (I will show that later).

In general, the number of distinct users is 339,231, while the number of distinct items that have been rated is 83,045.

Back to the ratings dataset, the summary already give us some **standard descriptive statistics**: the `min` and `max` functions let us know that the ratings go from 1 to 5; the `mean` equal to 4.244 tells us that, in general, the users have rated the items with very high grades, mostly with 5; also the low `standard deviation` indicates that lots of rating values are close to the mean.

This observation is also confirmed by the `quartiles` and the `median`. Indeed, the first quartile is already equal to 4, which is a really high number for data that takes values up to 5 - since it represents the middle number between the minimum value and the median. The same observation can be done for the median, that is equal to 5, i.e. the maximum value.

The information about timestamps given by the summary shows that the ratings are related to the years 1998 to 2014, but the majority is between 2011 and 2014.

After I had an overview of data, I immediately check for the **presence of NA values**, that could create problems when computing statistics on data and that, if present, have to be substituted with real values or omitted.

Code

```
The ratings dataset does not contain NA values
```

Code

```
The metadata dataset does contain NA values
```

As I already mentioned before, while the ratings dataset does not contain NA values, the metadata dataset contains NA values, but in order to find all of them, we need to search also inside the lists of some attribute. To do so, I will check only the attributes that could be useful for the construction of a recommendation system (for example, I discard the *description* in advance).

Code

```
The attribute asin does not contain NA values
```

Code

```
The attribute Musical Instruments salesRank does contain NA values -> 21714 over 8490
1
```

Code

```
The attribute related does contain NA values -> 188242 over 339604
```

Code

```
The attribute brand does contain NA values -> 50387 over 84901
```

Code

```
The attribute price does contain NA values -> 15831 over 84901
```

Unfortunately, the number of NA values for *Musical Instruments sales rank*, *related* and *brand* is too high: in the first attribute a quarter of the values are missing, while in the other two more than an half of the values are NA. For this reason, it would be pointless to try to substitute the NA values with the mean of the present values, or to delete the data with NA values from the dataset. Therefore, I think in this case it is appropriate not to consider these attributes for the development of the recommendation systems.

However, as regards the number of NA values for the attribute *price*, it is borderline and I will decide later if it could be useful to analyse it.

To be shure that I can use all the items for the construction of my recommendation system, I check if all of them contain the category “Musical Instruments” among the categories.

Code

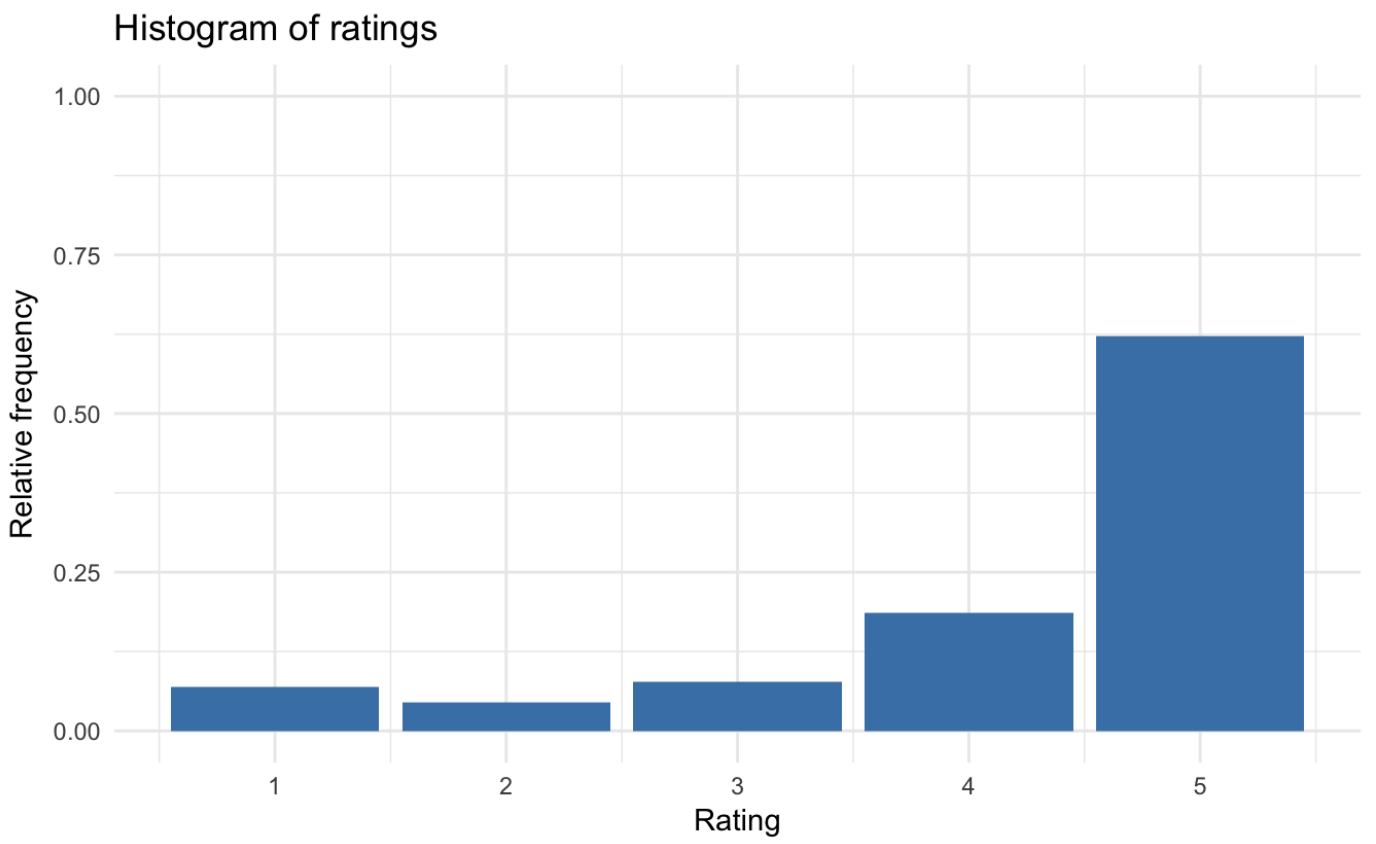
```
Items of the dataframe that have 'Musical Instruments' in their categories: 84901 ove
r 84901
```

# Representation and visualization of data

In order to explore and describe data, I also use some **graphic representations**, that can show the data distribution and some of its characteristics.

I start by blotting an histogram showing the relative frequencies of ratings.

Code



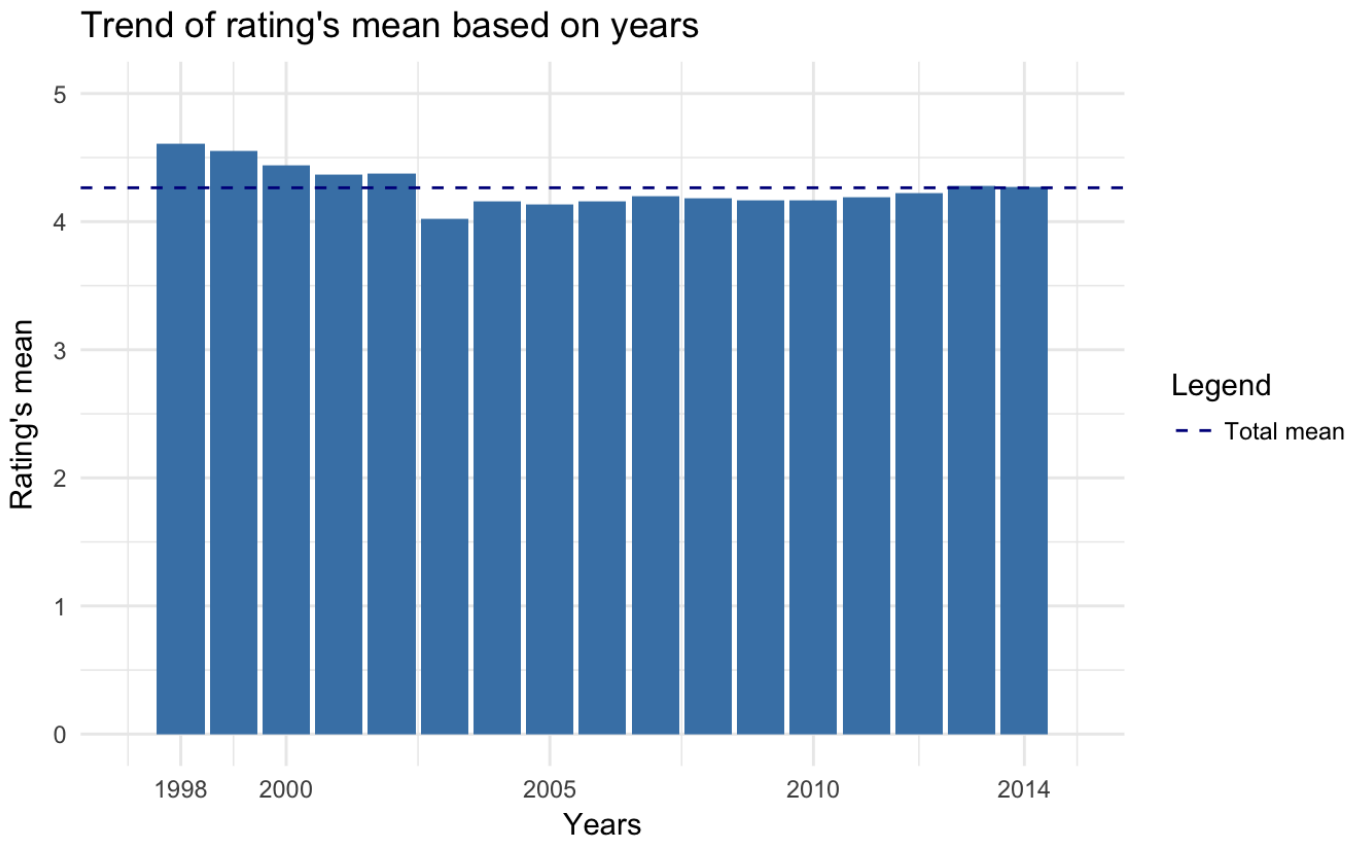

Code

```
    1     2     3     4     5 
7.0%  4.5%  7.7% 18.7% 62.1% 
```

The figure shows that there is a large majority of high ratings: more than a half of the ratings are equal to 5. This confirms the previous considerations resulting from the descriptive statistics.

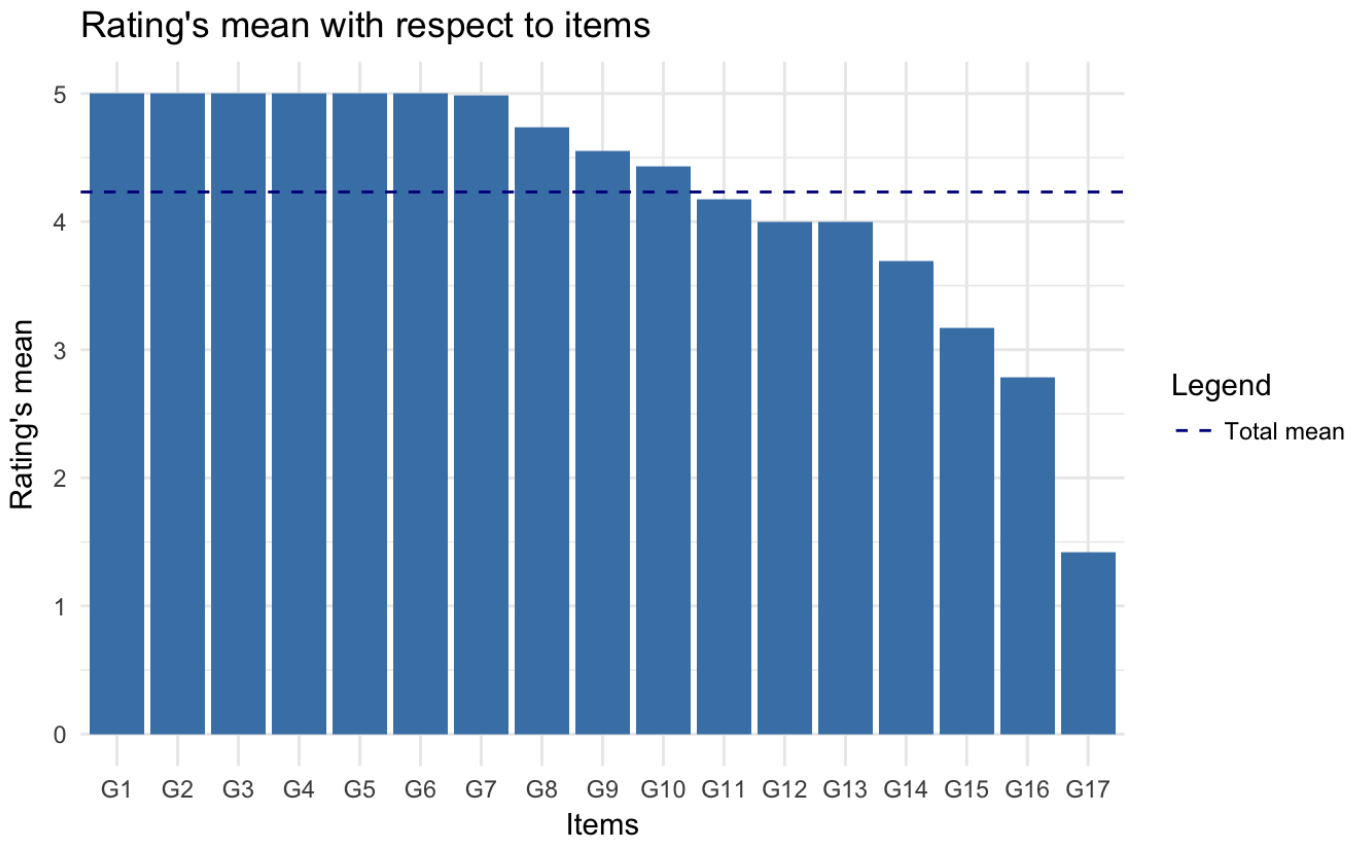
The following plot shows the trend of the rating's mean as the years passed.

Code




Since the number of distinct items that have been rated is very big (83,045), in order to visualize the distribution of rating with respect to the items I decided to represent the items' ratings in the plot by 17 ordered groups of 4885 items each, so that a clearer graph is obtained. The figure shows the rating, aggregated by mean, for each group of items.

Code



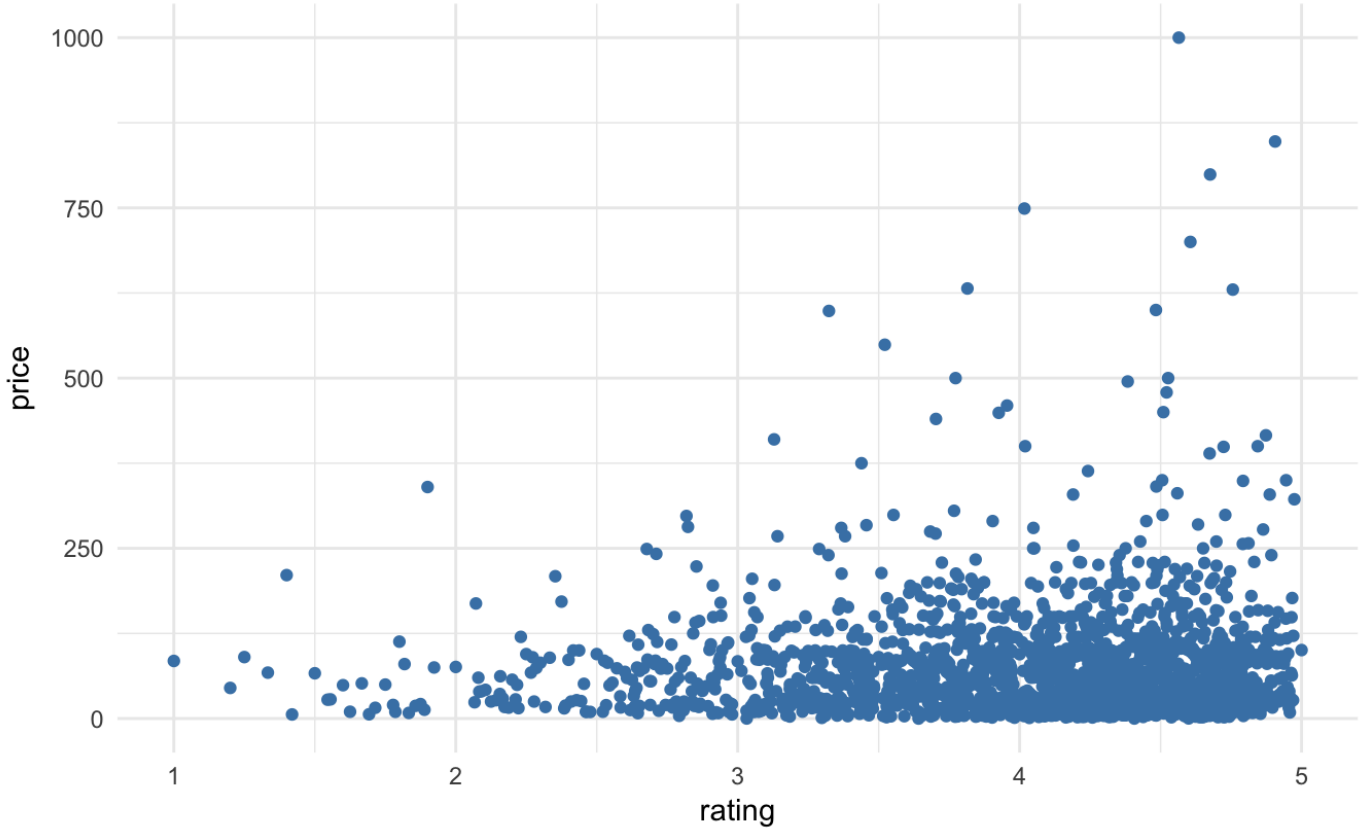

It could be also interesting to see the relation between the ratings and the prices of items. To obtain it, I discard from the metadata dataset all the items that haven't been rated and from both the datasets all the items that have the price value equal to NA. Then, I compute the correlation matrix between the ratings and the prices to check for the presence of a linear correlation between data and I also plot the distribution of the two related variable to have an overall view and to check for the presence of non-linear relations.

Hide

```
# -- Discard from metadata all the items that haven't been rated
metadata = metadata[is.element(metadata$asin, ds$item),]
# -- Discard from both the dataset all the items that have the price value equal to N
A
metadata_price = metadata[!is.na(metadata$price),]
metadata_price = data.frame("asin" = metadata_price$asin,
                                                    "price" = metadata_price$pric
e,
                                                    stringsAsFactors = FALSE)
metadata_price = metadata_price[order(metadata_price$asin),]
ds_price = ds[is.element(ds$item, metadata_price$asin),]
ds_price = aggregate(ds_price$rating, by = list(ds_price$item), mean)
ds_price = ds_price[order(ds_price$Group.1),]
ds_price = data.frame("item" = ds_price$Group.1,
                                            "price" = metadata_price$price,
                                            "rating" = ds_price$x,
                                            stringsAsFactors = FALSE)
means = aggregate(ds_price$price, by = list(ds_price$rating), mean)
names(means) = c("rating", "price")
cormat = round(cor(means), 4)
cormat
```

```
       rating  price
rating 1.0000 0.0097
price  0.0097 1.0000
```

Code




The resulting correlation matrix shows that there is no linear correlation between the variables *rating* and *price*, as the correlation value is very close to zero. However, from the graph we can deduct the presence of a non-linear relation between the two variables: as the ratings grows, also the prices grows, but it is also true that a lot of items with high rating have a very low price and the higher the rating, the denser is the “cloud” of low prices.

# Pre-processing of data

In order to create the recommendation systems, I will use only the ratings dataset, because the metadata dataset contains too many NA values. I remove the attribute *timestamp* from the ratings dataset since I don’t need it.

Hide

```
rating = ds[,c("user", "item", "rating")]
```

I will use a **subset of users and items** to create the recommendation systems for two reasons:

- Using all the data would be very expensive in terms of computational cost, therefore, reducing data dimensionality, I can speed up the computation;
- Most of the users appear in only one rating, so I will keep only the users that appear in 6 or more ratings in order to reduce the sparsity of the utility matrix; for the same reason, I will keep only the items that have been rated 20 or more times.

Hide

```
# -- Check if each user rated each items only once
# length(unique(c(ds$user, ds$item)))           # number of distinct couples "user, i
tem" equals number of distinct users
old_nrow = nrow(rating)
# -- Mean of the number of ratings received by each item
item_rating = aggregate(rating$rating, by = list(rating$item), length)
names(item_rating) = c("item", "nrating")
item_mean = mean(item_rating$nrating)           # item_mean = 6.023
old_nrow_item = nrow(item_rating)
# -- Items filtering
item_rating = item_rating[item_rating$nrating >= 20,]
rating = rating[is.element(rating$item, item_rating$item),]
# -- Mean of the number of ratings done by each user
user_rating = aggregate(rating$rating, by = list(rating$user), length)
names(user_rating) = c("user", "nrating")
user_mean = mean(user_rating$nrating)               # user_mean = 1.47
old_nrow_user = nrow(user_rating)
# -- Users filtering
user_rating = user_rating[user_rating$nrating >= 6,]
rating = rating[is.element(rating$user, user_rating$user),]
cat("Subset of rating -> ", "Total number of ratings:", nrow(rating), "over", old_nro
w, "\n",
        "\t\t\t\t\t", "Number of distinct users:", length(unique(rating$user)), "ove
r", old_nrow_user, "\n",
        "\t\t\t\t\t", "Number of distinct items:", length(unique(rating$item)), "ove
r", old_nrow_item, "\n")
```

```
Subset of rating ->  Total number of ratings: 15864 over 500175
                     Number of distinct users: 1840 over 206538
                     Number of distinct items: 3147 over 83045
```

The resulting dataset contains 15,864 ratings, 1,840 distinct users and 3,147 distinct items. This will be the dataset on which I will work from now on.

# Recommendation systems

In order to build the two recommendation systems required by the project, I'm going to use **collaborative filtering** (CF) in both cases: one of them will make use of the *user-based collaborative filtering* ( `UBCF` ), the other will make use of the *item-based collaborative filtering* ( `IBCF` ).

Collaborative filtering is an algorithm that uses given rating data by many users for many items as the basis for predicting missing ratings or for creating a list of items with the top-N items to recommend to a given user, called the active user. The data are organized in a $m \times n$ user-item matrix, called *utility matrix* or *sparse matrix*, where each row represents a user and each column represents an item. Into each user-item cell it is stored the rating of that user for that item. Usually, the utility matrix is very sparse, because only a small fraction of ratings are known, so most of the cells of the matrix contain NA values.

Two of the main categories of CF algoritms are the user-based collaborative filtering and the item-based collaborative filtering. The `UBCF` algorithms bases the recommendations on the similarity between users: the assumption is that users with similar preferences will rate items similarly. Each user is represented by his row of ratings in the utility matrix, so, in order to compute the similarity between users, we need to apply a similarity measure (for istance, the *cosine similarity*) between the row of the active user and all the other rows. In this way, we can find the most similar users and use their ratings in two ways:

- The list of best rated items of the similar users can be used to make recommendations to the active user;
- The missing rating of the active user can be predicted by aggregating the ratings of the similar users to form a prediction.

The `IBCF` algorithms bases the recommendations on the similarity between items: the assumption is that users will prefer items that are similar to other items they like. Each item is represented by his column of ratings in the utility matrix, so, in order to compute the similarity between items, we need to apply a similarity measure (for istance, the *cosine similarity*) between all the columns of the utility matrix. In this way, we can build a $n \times n$ item-item similarity matrix containing the similarity values for each items couple. To reduce the dimensionality of the similarity matrix, instead we could build a $n \times k$ item-item similarity matrix containing, for each item, the similarity values of only the k most similar items. From the similarity matrix, we can use the items that are the most similar to the best rated items of the active user in order to obtain:

- A list of similar items to make recommendations to the active user;
- The missing rating of the active user by aggregating the ratings of the similar items to form a prediction.

In order to build the two recomendation systems, I'm going to use the *recommenderlab* library, which provides several algorithms for this tipe of task that are already implemented.

First, I need to create the **utility matrix** from the ratings dataset. The function `dcast` is perfect for this purpose: it simply stores the users as rows' indices, the items as columns' indices and the ratings as values inside the user-item cells.

Hide

```
utility_matrix = dcast(rating, formula = user~item, value.var = 'rating')
```

Code

```
Number of real rating values: 15864 over 5790480
```

The utility matrix contains 5,790,480 values, of which only 15,864 are real ratings and the others are missing values (NA).

Before working on the utility matrix, I need to convert it into `matrix` type (the function `dcast` returns a list) in order to be able to transform it in a `realRatingMatrix` to handle it through the *recommentderlab* library.

Hide

```
m = as.matrix(utility_matrix[,-1])
rownames(m) = utility_matrix[,1]
# print(m[1:10,1:8])
utility_m <- as(m, "realRatingMatrix")
utility_m
```

```
1840 x 3147 rating matrix of class 'realRatingMatrix' with 15864 ratings.
```

Before building the actual recommendation systems, I create an **evaluation scheme** that determines what and how data is used for training and testing and that allows me to evaluate the two models at the end. The division in training set and test set makes it possible to predict the ratings of the users in the test set using the training set. Through the evaluation scheme, I split the dataset using 75% of users for the training set and 25% for the test set, I consider good ratings only the ratings equal to 5 (anyway, more than a half of the ratings are equal to 5) and I set to 4 the value of given ratings, so that in the test set each user have 4 ratings given and the recommendation systems have to predict the others that are missing.

Hide

```
e = evaluationScheme(utility_m, method = "split", train = 0.75, given = 4, goodRating
 = 5)
e
```

```
Evaluation scheme with 4 items given
Method: 'split' with 1 run(s).
Training set proportion: 0.750
Good ratings: >=5.000000
Data set: 1840 x 3147 rating matrix of class 'realRatingMatrix' with 15864 ratings.
```

After that, I create the two recommendation systems with the function `Recommender`, where I can set the training set as the data that it has to use to build the recommendation system and the method, depending on whether it has to use the `UBCF` or the `IBCF`. Then, I use the created `Recommenders` in order to predict the missing ratings in the test set through the function `predict`.

Hide

```
# --- Recommendation system using UBCF
rUBCF = Recommender(getData(e, "train"), method = "UBCF")
rUBCF
```

```
Recommender of type 'UBCF' for 'realRatingMatrix'
learned using 1380 users.
```

Hide

```
pUBCF = predict(rUBCF, getData(e, "known"), type = "ratings")
pUBCF
```

```
460 x 3147 rating matrix of class 'realRatingMatrix' with 902041 ratings.
```

Hide

```
# --- Recommendation system using IBCF
rIBCF = Recommender(getData(e, "train"), method = "IBCF")
rIBCF
```

```
Recommender of type 'IBCF' for 'realRatingMatrix'
learned using 1380 users.
```

Hide

```
pIBCF = predict(rIBCF, getData(e, "known"), type = "ratings")
pIBCF
```

```
460 x 3147 rating matrix of class 'realRatingMatrix' with 57147 ratings.
```

Finally, I evaluate the predictions of the two recommendation systems by comparing their accuracy through an evaluation matrix. The matrix contains, for each model, the *root-mean-square error (RMSE)*, that is the square root of the mean square error, the *mean squared error (MSE)*, that measures the average of the squares of the

differences between predicted values and observed values, and the *mean absolute error (MAE)*, that represents the average absolute difference between predicted values and observed values.

Hide

```
# Error between prediction and unknown part of the test data
error = rbind(UBCF = calcPredictionAccuracy(pUBCF, getData(e, "unknown")),
                           IBCF = calcPredictionAccuracy(pIBCF, getData(e, "unknown"
)))
error
```

```
          RMSE       MSE       MAE
UBCF 0.9631215 0.9276031 0.6462931
IBCF 1.3906379 1.9338738 0.9206462
```

I tried different configurations for the data filtering, lowering or rasing the number of users and items selected to form the dataset: obviously, the higher the minimum number of ratings for each user and item, the lower the error of the predictions. In general, the `UBCF` works better than the `IBCF` with every configuration I tried, since all the three measures of error are higher in the `UBCF` than in the `IBCF`.