

Predicting book ratings: can a simple model outperform singular value decomposition?

Nir Levy

1/14/2022

Harvardx Data Science - ‘Choose Your Own Project’ Report

Nir Levy, January 2022

Summary

This report describes my work on the project of my choice, within the Capstone course of Harvardx’s Data Science Professional Certificate.

The question that I investigate is: can a simple model that predicts book ratings perform better than more advanced ones, that apply singular value decomposition?

The question is interesting and may lead to useful learnings, since the simple models have important advantages in a real recommendation system: 1. They could easily be applied to new users and new books, thus providing a good way to deal with the ‘cold start’ problem. 2. If a user rates an item and we are interested in recommending a new item immediately, we do not need to train the algorithm on the whole matrix again. We can immediately apply a model that requires a simple calculation, thus saving both time and processing power.

The analysis was structured as follows:

Section 1 - Downloading the data

In order to address this question, I downloaded the “Book-Crossing” dataset that contains book ratings provided by users of the Book-Crossing website. The data is available online here (<http://www2.informatik.uni-freiburg.de/~cziegler/BX/>) and on Kaggle (<https://www.kaggle.com/somnambwl/bookcrossing-dataset>). The website’s url is: <https://www.bookcrossing.com/>.

Section 2 - Preparing the data

Next, I prepared the data for analysis.

Section 3 - Exploratory data analysis

Afterwards I explored the data through some calculations and visualizations.

Section 4 - Evaluating simple models (validation part 1)

Afterwards I evaluated some simple models, similar to those that we learned in the machine learning course. However, I introduced a bit more information to the models. While in the course we only included unique user effects and unique item effects, I included ‘side information’ as well. That is, I included information on users, such as age and country, and information on the items (books), such as author and publisher.

Section 5 - Introducing regularization (validation part 2)

I then added regularization to the simple models, to account for the difference between average ratings provided by a few users and average ratings provided by many users.

Section 6 - Evaluating more advanced models (validation part 3)

I proceeded to evaluate some more advanced ones, through validation. First I evaluated the ‘Popular’ model because of its’ simplicity. Then I evaluated the ‘Singular Value Decomposition’ (SVD) and ‘Funk Singular Value Decomposition’ models (SVDF), since the matrix was very sparse. I discovered that the simple models produced a much lower Root Mean Squared Error (RMSE) than the advanced ones!

Section 7 - Applying the chosen model to the test set

Finally, I applied the model that produced the lowest RMSE to the test dataset. It produced an RMSE that was very similar to the one that it produced in the validation stage, indicating that the model was probably not over-fitted in the validation stage. This was expected, given the low number of features in the model and its’ simplicity.

Section 8 - Conclusion and discussion

My conclusion is that under certain conditions, simple models can outperform more advanced ones. There could be several reasons for this, in this case. First, the dataset was rather small. This could be problematic for algorithms that rely on the associations between ratings given by users (e.g. two user give similar ratings) and the associations of ratings given to items (e.g. two books receive similar ratings). Third, the matrix was very sparse, and that could be too difficult, even for the advanced algorithms. Finally, I may have made a mistake in the analysis, even though I tried not to do that :). Further analyses would be required in order to find out what combination of these possibilities is indeed the cause of the poor performance of the advanced models in comparison to the simple ones.

Personally, I feel like I learned an important lesson - sometimes there is no tradeoff between simplicity and accuracy. ‘Simple’ could sometimes be more accurate!

Table of contents

1. Downloading the data
2. Preparing the data
3. Exploratory data analysis
4. Evaluating simple models (validation part 1)
5. Introducing regularization (validation part 2)
6. Evaluating more advanced models (validation part 3)
7. Applying the chosen model to the test set
8. Conclusion and discussion

1. Downloading the data

Let us begin by downloading the data.

```
#####  
### 1. Downloading the data ###  
#####  
  
### Installing packages  
suppressMessages(if(!require(downloader)) install.packages("downloader", repos = "http://cran.us.r-project.org"))  
  
## Warning: package 'downloader' was built under R version 4.0.5  
  
suppressMessages(if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org"))  
  
## Warning: package 'dplyr' was built under R version 4.0.5  
  
suppressMessages(if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org"))  
  
## Warning: package 'tidyverse' was built under R version 4.0.5  
  
## Warning: package 'ggplot2' was built under R version 4.0.5  
  
## Warning: package 'tibble' was built under R version 4.0.5  
  
## Warning: package 'tidyr' was built under R version 4.0.5  
  
## Warning: package 'readr' was built under R version 4.0.5  
  
## Warning: package 'purrr' was built under R version 4.0.5  
  
## Warning: package 'stringr' was built under R version 4.0.5  
  
## Warning: package 'forcats' was built under R version 4.0.5  
  
suppressMessages(if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org"))  
  
## Warning: package 'caret' was built under R version 4.0.5  
  
suppressMessages(if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org"))  
  
## Warning: package 'data.table' was built under R version 4.0.5  
  
suppressMessages(if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org"))  
suppressMessages(if(!require(recommenderlab)) install.packages("recommenderlab", repos = "http://cran.us.r-project.org"))  
  
## Warning: package 'recommenderlab' was built under R version 4.0.5  
  
## Warning: package 'Matrix' was built under R version 4.0.5  
  
## Warning: package 'arules' was built under R version 4.0.5  
  
## Warning: package 'proxy' was built under R version 4.0.5
```

```

suppressMessages(if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org"))

### loading libraries
suppressWarnings(suppressMessages(library(downloader)))
suppressWarnings(suppressMessages(library(dplyr)))
suppressWarnings(suppressMessages(library(tidyverse)))
suppressWarnings(suppressMessages(library(caret)))
suppressWarnings(suppressMessages(library(data.table)))
suppressWarnings(suppressMessages(library(ggplot2)))
suppressWarnings(suppressMessages(library(recommenderlab)))
suppressWarnings(suppressMessages(library(stringr)))

### description of the data set
### by publisher: "http://www2.informatik.uni-freiburg.de/~ctiegle/BX/"
### in Kaggle: "https://www.kaggle.com/somnambwl/bookcrossing-dataset"
### (the website: "https://www.bookcrossing.com/")

### downloading the dataset
download("http://www2.informatik.uni-freiburg.de/~ctiegle/BX/BX-CSV-Dump.zip", dest="dataset.zip", mode="wb")
unzip("dataset.zip")

### converting the files to data frame format
books<-read.csv(file = 'BX-Books.csv', sep=";")
users<-read.csv(file = 'BX-Users.csv', sep=";")
ratings<-read.csv(file = 'BX-Book-Ratings.csv', sep=";")

### saving the data frames
saveRDS(books, "books")
saveRDS(users, "users")
saveRDS(ratings, "ratings")

```

2. Preparing the data

Let us now prepare the data for the analysis. Our preparation includes creating an additional variable that will enable us to include interaction effects between users and authors (it seems reasonable that a users' rating of a book by a certain author would be rather similar to their rating of other books by the same author). We also create another variable, 'age bracket', in order to account for non-linear effects of age on our outcome in the models that we will apply. We also perform 'sanity checks' by examining some key distributions.

```

#####
### 2. Preparing the data ###
#####

### cleaning the working space
rm(list=ls())

### loading the data
books<-readRDS("books")
users<-readRDS("users")
ratings<-readRDS("ratings")

```

```
### exploring the three data files
```

```
dim(books)
```

```
## [1] 115253      8
```

```
names(books)
```

```
## [1] "ISBN"           "Book.Title"       "Book.Author"
## [4] "Year.Of.Publication" "Publisher"         "Image.URL.S"
## [7] "Image.URL.M"      "Image.URL.L"
```

```
dim(users)
```

```
## [1] 140291      3
```

```
names(users)
```

```
## [1] "User.ID" "Location" "Age"
```

```
head(users)
```

```
##   User.ID           Location Age
## 1      1      nyc, new york, usa NULL
## 2      2 stockton, california, usa  18
## 3      3 moscow, yukon territory, russia NULL
## 4      4      porto, v.n.gaia, portugal  17
## 5      5 farnborough, hants, united kingdom NULL
## 6      6      santa monica, california, usa  61
```

```
dim(ratings)
```

```
## [1] 493813      3
```

```
names(ratings)
```

```
## [1] "User.ID" "ISBN"      "Book.Rating"
```

```
head(ratings)
```

```
##   User.ID      ISBN Book.Rating
## 1  276725 034545104X          0
## 2  276726 0155061224          5
## 3  276727 0446520802          0
## 4  276729 052165615X          3
## 5  276729 0521795028          6
## 6  276733 2080674722          0
```

```
### extracting country names from the 'users' data frame
users_processed<-users # duplicating the raw 'users' dataset
```

```
### examining the first 10 values in the users$Location column
users$Location[1:10]
```

```
## [1] "nyc, new york, usa"          "stockton, california, usa"
## [3] "moscow, yukon territory, russia" "porto, v.n.gaia, portugal"
## [5] "farnborough, hants, united kingdom" "santa monica, california, usa"
## [7] "washington, dc, usa"          "timmins, ontario, canada"
## [9] "germantown, tennessee, usa"    "albacete, wisconsin, spain"
```

```
### The users$Location column contains three entries, separated by commas.
### In our model we will only use the country
```

```
### Extracting the country names
### extracting country names
users_processed$country<-sub(".*, ", "", users_processed$Location)

### making sure that it worked
### first 10 rows
users_processed$country[1:10]
```

```
## [1] "usa"          "usa"          "russia"       "portugal"
## [5] "united kingdom" "usa"         "usa"         "canada"
## [9] "usa"          "spain"
```

```
users_processed$Location[1:10]
```

```
## [1] "nyc, new york, usa"          "stockton, california, usa"
## [3] "moscow, yukon territory, russia" "porto, v.n.gaia, portugal"
## [5] "farnborough, hants, united kingdom" "santa monica, california, usa"
## [7] "washington, dc, usa"          "timmins, ontario, canada"
## [9] "germantown, tennessee, usa"    "albacete, wisconsin, spain"
```

```
### an arbitrary sample of rows
users_processed$country[647:658]
```

```
## [1] "australia"      "canada"        "usa"           "usa"
## [5] "usa"           "usa"           "united kingdom" "canada"
## [9] "usa"           "new zealand"   "usa"           "usa"
```

```
users_processed$Location[647:658]
```

```
## [1] "perth, western australia, australia" "kamloops, british columbia, canada"
## [3] "san diego, california, usa"          "santa cruz, california, usa"
## [5] "interlachen, florida, usa"           "slippery rock, pennsylvania, usa"
## [7] "bristol, wales, united kingdom"      "ottawa, ontario, canada"
## [9] "monroe, north carolina, usa"          "feilding, manawatu, new zealand"
## [11] "chattanooga, tennessee, usa"        "jasper, georgia, usa"
```

```
names(users_processed)
```

```
## [1] "User.ID" "Location" "Age" "country"
```

```
### merging all three datasets to one file, on the rating level  
### i.e. each row will contain a rating, and information  
### about both the user and the book that were involved
```

```
### adding user information
```

```
names(users_processed)
```

```
## [1] "User.ID" "Location" "Age" "country"
```

```
class(users_processed$User.ID)
```

```
## [1] "character"
```

```
class(users_processed$User.ID)
```

```
## [1] "character"
```

```
### converting the users$User.ID column to numeric form, since that is its' form in the ratings dataset  
suppressWarnings(users_processed$User.ID<-as.numeric(users_processed$User.ID))
```

```
### merging the files
```

```
all_together <- left_join(ratings, users_processed, by = "User.ID")
```

```
### making sure that the merged file is fine
```

```
head(all_together)
```

```
##   User.ID      ISBN Book.Rating      Location Age country  
## 1  276725 034545104X          0      tyler, texas, usa NULL    usa  
## 2  276726 0155061224          5      seattle, washington, usa NULL    usa  
## 3  276727 0446520802          0 h, new south wales, australia 16 australia  
## 4  276729 052165615X          3      rijeka, n/a, croatia 16  croatia  
## 5  276729 0521795028          6      rijeka, n/a, croatia 16  croatia  
## 6  276733 2080674722          0      paris, n/a, france 37   france
```

```
dim(all_together)
```

```
## [1] 493813      6
```

```
### adding books information
```

```
names(books)
```

```
## [1] "ISBN" "Book.Title" "Book.Author"  
## [4] "Year.Of.Publication" "Publisher" "Image.URL.S"  
## [7] "Image.URL.M" "Image.URL.L"
```

```
class(books$ISBN)
```

```
## [1] "character"
```

```
class(ratings$ISBN)
```

```
## [1] "character"
```

```
### merging the files
```

```
all_together <- left_join(all_together, books, by = "ISBN")
```

```
### making sure that the merged file is fine
```

```
names(all_together)
```

```
## [1] "User.ID"          "ISBN"              "Book.Rating"
## [4] "Location"         "Age"               "country"
## [7] "Book.Title"       "Book.Author"       "Year.Of.Publication"
## [10] "Publisher"        "Image.URL.S"       "Image.URL.M"
## [13] "Image.URL.L"
```

```
dim(all_together)
```

```
## [1] 493813    13
```

```
### keeping the relevant columns only
```

```
colnames<-names(all_together)
```

```
colnames
```

```
## [1] "User.ID"          "ISBN"              "Book.Rating"
## [4] "Location"         "Age"               "country"
## [7] "Book.Title"       "Book.Author"       "Year.Of.Publication"
## [10] "Publisher"        "Image.URL.S"       "Image.URL.M"
## [13] "Image.URL.L"
```

```
keep<-c(colnames[1], colnames[2], colnames[3], colnames[5], colnames[6], colnames[8], colnames[9], colnames[10], colnames[11], colnames[12], colnames[13])
```

```
dat<-all_together[keep]
```

```
head(dat)
```

```
##   User.ID      ISBN Book.Rating Age  country      Book.Author
## 1  276725 034545104X          0 NULL     usa      M. J. Rose
## 2  276726 0155061224          5 NULL     usa             <NA>
## 3  276727 0446520802          0  16 australia             <NA>
## 4  276729 052165615X          3  16  croatia             <NA>
## 5  276729 0521795028          6  16  croatia             <NA>
## 6  276733 2080674722          0  37   france Michel Houellebecq
##   Year.Of.Publication      Publisher
## 1                   2002 Ballantine Books
## 2                   <NA>             <NA>
## 3                   <NA>             <NA>
## 4                   <NA>             <NA>
## 5                   <NA>             <NA>
## 6                   1998      Flammarion
```



```
### simplifying the names
names(dat)<- c("userid", "bookid", "rating", "age", "country", "author", "year", "publisher")
names(dat) # making sure that the renaming worked properly
```

```
## [1] "userid"      "bookid"      "rating"      "age"         "country"     "author"
## [7] "year"        "publisher"
```

```
### checking the classes of the columns that are supposed to be numeric
class(dat$userid)
```

```
## [1] "numeric"
```

```
class(dat$bookid)
```

```
## [1] "character"
```

```
class(dat$rating)
```

```
## [1] "integer"
```

```
class(dat$age)
```

```
## [1] "character"
```

```
class(dat$year)
```

```
## [1] "character"
```

```
### converting columns that should be numeric to numeric class
dat_backup<-dat # creating a backup
saveRDS(dat_backup, "dat_backup") # saving it
```

```
### converting to numeric
suppressWarnings(dat$bookid<-as.numeric(dat$bookid))
suppressWarnings(dat$age<-as.numeric(dat$age))
suppressWarnings(dat$year<-as.numeric(dat$year))
```

```
### making sure that it worked
class(dat$userid)
```

```
## [1] "numeric"
```

```
class(dat$bookid)
```

```
## [1] "numeric"
```

```

class(dat$rating)

## [1] "integer"

class(dat$age)

## [1] "numeric"

class(dat$year)

## [1] "numeric"

### preparing a column with the concatenation of userid and author
### (see the rationale in the beginning of this section)

### first, let's check for missing values
n_missing_userid<-sum(is.na(dat$userid)) # the number of missing values in the userid column
n_missing_userid

## [1] 0

n_missing_author<-sum(is.na(dat$author)) # the number of missing values in the author column
n_missing_author

## [1] 294664

nrow(dat)

## [1] 493813

### the author columns contains missing values
### indexing the rows that do not have a missing value in the author column
index<-which(!is.na(dat$author))

### making sure that the indexing worked properly
head(index)

## [1] 1 6 11 12 13 14

length(index)+n_missing_author-nrow(dat) # this should be zero

## [1] 0

### concatenating userid and author in rows that contain both
dat$userid_author<-"missing" # creating default value
dat$userid_author[index]<-paste(dat$userid[index], dat$author[index]) # concatenating
head(dat$userid_author)

## [1] "276725 M. J. Rose"      "missing"
## [3] "missing"                "missing"
## [5] "missing"                "276733 Michel Houellebecq"

```

```
dat$userid_author[dat$userid_author=="missing"]<-NA # replacing "missing" with NAs
```

```
### making sure that it worked properly
dat$userid_author[4300:4400] # examining
```

```
##      [1] NA                      NA
##      [3] NA                      NA
##      [5] "278390 Patricia Cornwell" NA
##      [7] NA                      "278390 Kate White"
##      [9] "278390 Margaret Atwood"  NA
##     [11] "278390 Dan Brown"        "278390 Alice Walker"
##     [13] "278390 Alice Walker"     NA
##     [15] NA                      NA
##     [17] "278390 Amy Tan"          "278390 Amy Tan"
##     [19] NA                      NA
##     [21] NA                      "278400 Terry C. Johnston"
##     [23] NA                      NA
##     [25] "278409 Amy Ephron"       NA
##     [27] NA                      NA
##     [29] NA                      NA
##     [31] NA                      NA
##     [33] NA                      NA
##     [35] NA                      NA
##     [37] NA                      "278418 Virginia Hamilton"
##     [39] NA                      "278418 C. S. Lewis"
##     [41] NA                      "278418 Alan Paton"
##     [43] NA                      NA
##     [45] NA                      NA
##     [47] NA                      NA
##     [49] NA                      NA
##     [51] "278418 Ellen Weiss"     NA
##     [53] "278418 Richard Hefter"  "278418 Richard Hefter"
##     [55] NA                      NA
##     [57] NA                      NA
##     [59] "278418 Roseanne"       NA
##     [61] NA                      "278418 Olivia Goldsmith"
##     [63] NA                      NA
##     [65] NA                      "278418 Tony Hillerman"
##     [67] NA                      NA
##     [69] NA                      NA
##     [71] NA                      "278418 Margaret Wise Brown"
##     [73] NA                      NA
##     [75] NA                      NA
##     [77] NA                      "278418 Thacher Hurd"
##     [79] NA                      NA
##     [81] NA                      "278418 Michael Johnson"
##     [83] NA                      "278418 John Gunther"
##     [85] "278418 John F. Kennedy"  NA
##     [87] NA                      "278418 Bryan Burrough"
##     [89] "278418 Oscar Hijuelos"  NA
##     [91] NA                      NA
##     [93] NA                      NA
##     [95] NA                      "278418 Bill Dugan"
```

```
## [97] NA "278418 Barbara Delinsky"
## [99] "278418 Chri Carter" NA
## [101] NA
```

```
nonmissing1<-sum(!is.na(dat$userid_author)) # counting non-missing values
nonmissing2<-nrow(dat)-n_missing_author
nonmissing1-nonmissing2 # this should be zero
```

```
## [1] 0
```

```
### creating age brackets for the analysis
### (see rationale in the beginning of this section)
train$age<-round(train$age,0) # rounding the values of the age variable, in case they aren't rounded yet
```

```
## Error in train$age: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age<=10]<-"0-10"
```

```
## Error in 'tmp*$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=11 & train$age<=20]<-"11-20"
```

```
## Error in 'tmp*$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=21 & train$age<=30]<-"21-30"
```

```
## Error in 'tmp*$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=31 & train$age<=40]<-"31-40"
```

```
## Error in 'tmp*$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=41 & train$age<=50]<-"41-50"
```

```
## Error in 'tmp*$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=51 & train$age<=60]<-"51-60"
```

```
## Error in 'tmp*$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=61 & train$age<=70]<-"61-70"
```

```
## Error in 'tmp*$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=71 & train$age<=80]<-"71-80"
```

```
## Error in `*tmp*`$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=81 & train$age<=90]<-"81-90"
```

```
## Error in `*tmp*`$age_bracket: object of type 'closure' is not subsettable
```

```
train$age_bracket[train$age>=91 & train$age<=100]<-"91-100"
```

```
## Error in `*tmp*`$age_bracket: object of type 'closure' is not subsettable
```

```
### examining the distribution of ratings  
table(dat$rating)
```

```
##  
##      0      1      2      3      4      5      6      7      8      9     10  
## 317794   658   1184   2622   3802  21546  15222  31300  42046  26549  31090
```

```
### there seem to be many rows with rating = 0  
### in the book crossing website, books are rated from 1 to 10  
### with an option to mark "haven't read" as well  
### (see the website: "https://www.bookcrossing.com/")  
### so we need to remove the rows with ratings of zero  
  
### removing the rows with ratings of zero  
dim(dat)
```

```
## [1] 493813      9
```

```
dat<-dat[dat$rating!=0,]  
dim(dat)
```

```
## [1] 176019      9
```

```
### saving  
saveRDS(dat, file="dat")  
  
### cleaning up the working space  
rm(list=ls())  
  
### reloading the dataset  
dat<-readRDS("dat")  
  
### cleaning memory  
invisible(gc())  
  
### checking the percentage of missing values in each variable
```

```
# creating a function that computes the percentage of missing values in a vector
percent_missing<-function(x){
  n_missing<-sum(is.na(x)) # calculate the number of missing values
  p_missing<-n_missing/length(x) # calculate the percentage of missing values
  p_missing # print the percentage
}
```

```
# applying the function to each of the columns
apply(dat, MARGIN = 2, percent_missing)
```

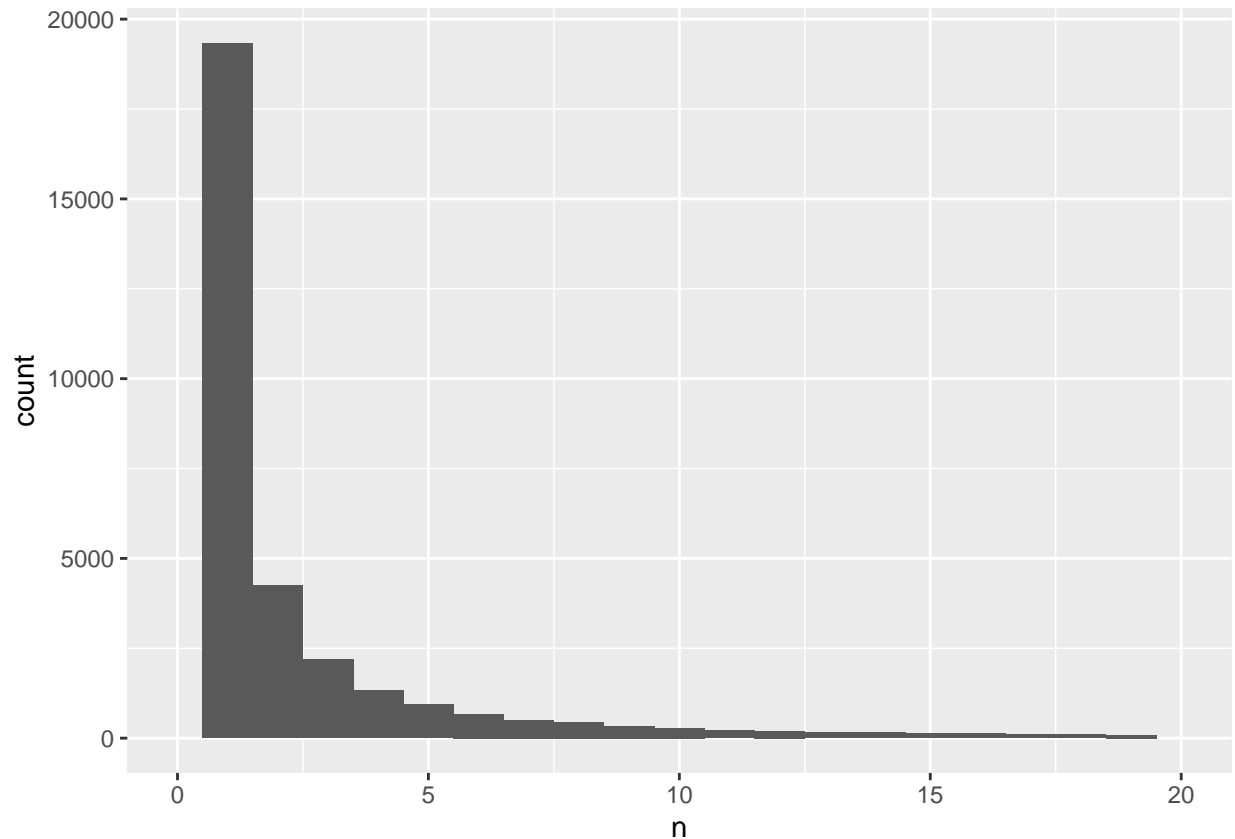
```
##      userid      bookid      rating      age      country
## 0.0000000 0.0828206 0.0000000 0.6116896 0.4580926
##      author      year      publisher userid_author
## 0.5797726 0.5798124 0.5797726 0.5797726
```

```
# examining the distribution of the number of ratings per user
ratings_per_user<-dat %>%
  filter(!is.na(rating)) %>%
  count(userid)

ggplot(ratings_per_user, aes(x=n)) +
  geom_histogram(binwidth=1) +
  xlim(0,20)
```

```
## Warning: Removed 1474 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



```
### splitting the data into training and test sets
### since the number of ratings per user is fairly low
### and some variables have a high percentage of missing values,
### I choose a 50-50 initial split to make sure that the test set is large enough.
### In the final split, the percentage of users in the training set will be larger than 50
### (and less than 50 in the test set) since we are going to remove users who appear only in the test s
### from the test set and add them to the training set.
```

```
# creating index of 50%
suppressWarnings(set.seed(1, sample.kind="Rounding")) # setting the seed
test_index <- createDataPartition(y = dat$rating, times = 1, p = 0.5, list = FALSE) # defining a 50% sp
train <- dat[-test_index,]
temp <- dat[test_index,]

# making sure that the temporary split is 50-50
length(test_index)/nrow(dat)
```

```
## [1] 0.5000028
```

```
# creating a test set that includes only userids and bookids that are also in the training set
test <- temp %>%
  semi_join(train, by = "bookid") %>%
  semi_join(train, by = "userid")

# adding rows that were included in the index, but not in the test set, to the training set
removed <- anti_join(temp, test)
```

```
## Joining, by = c("userid", "bookid", "rating", "age", "country", "author", "year", "publisher", "user")
```

```
train <- rbind(train, removed)
```

```
### making sure that the number of rows in the training and test sets is equal to  
### the number of rows in the full dataset  
nrow(dat)-(nrow(train)+nrow(test)) # this should be zero
```

```
## [1] 0
```

```
### checking the final relative sizes of the training and test sets  
nrow(train)/nrow(dat) # the train set is ~70% of the full data
```

```
## [1] 0.7740471
```

```
nrow(test)/nrow(dat) # the test set is ~30% of the full data
```

```
## [1] 0.2259529
```

```
### saving the files  
saveRDS(train, file="train")  
saveRDS(test, file="test")  
  
### cleaning the working space  
rm(list=ls())  
  
### cleaning memory  
invisible(gc())  
  
### reloading the training set  
train<-readRDS("train")
```

3. Exploring the data

Now we explore the data. We also ‘clean’ it where it seems to make sense, by removing age values that are above 100.

```
#####  
### 3. Exploring the data ###  
#####  
  
### we already examined the percentage of missing values in each column  
### and the distribution of the number of users above, in the full data set,  
### in order to determine the split between the training and test sets.  
### from this point until the final validation, we will proceed with the train data only.  
### let us examine some properties of the train data.  
  
### examining the number of unique values in each column  
dim(train)
```



```
## [1] 136247      9
```

```
names(train)
```

```
## [1] "userid"      "bookid"      "rating"      "age"
## [5] "country"     "author"      "year"        "publisher"
## [9] "userid_author"
```

```
### examining the number of unique users
n_distinct(train$userid)
```

```
## [1] 33095
```

```
nrow(train)
```

```
## [1] 136247
```

```
### examining the number of unique books
n_distinct(train$bookid)
```

```
## [1] 87817
```

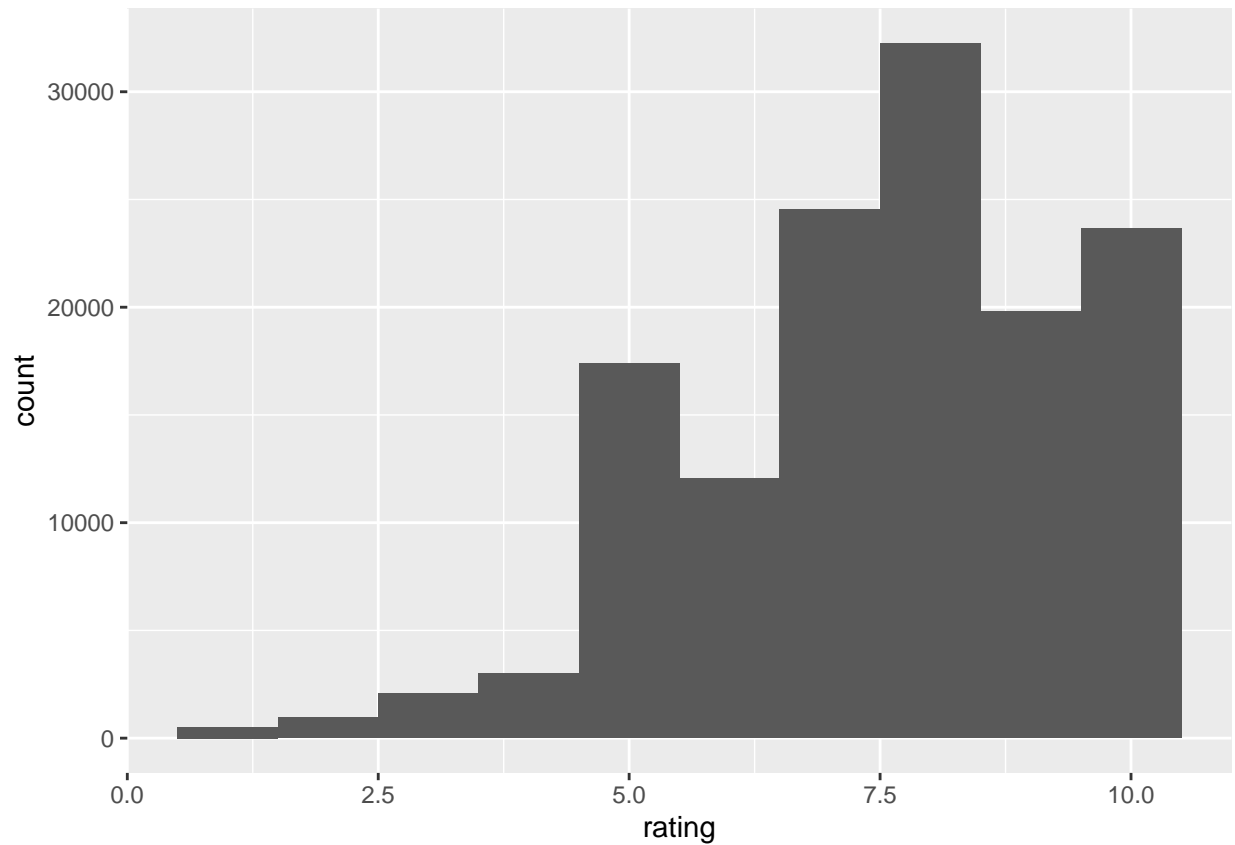
```
nrow(train)
```

```
## [1] 136247
```

```
### examining the distribution of ratings
table(train$rating)
```

```
##
##      1      2      3      4      5      6      7      8      9     10
##   517   943  2078  3008 17385 12058 24537 32246 19809 23666
```

```
train %>% ggplot(aes(x=rating)) +
  geom_histogram(binwidth=1)
```



```
### the distribution is skewed to the right.
```

```
### counting the countries  
n_distinct(train$country)
```

```
## [1] 218
```

```
### counting the authors  
n_distinct(train$author)
```

```
## [1] 18472
```

```
### counting the years  
n_distinct(train$year)
```

```
## [1] 82
```

```
### counting the publishers  
n_distinct(train$publisher)
```

```
## [1] 4391
```

```
### counting the usierid-author combinations
n_distinct(train$userid_author)
```

```
## [1] 50612
```

```
### checking the average number of times each value appears in each column
### creating the function
avpu_func<-function(x){
  x<-x[!is.na(x)] # removing missing values
  n_unique_units<-n_distinct(x) # counting distinct values
  n_values<-length(x) # counting the overall number of values
  avpu<-round(n_values/n_unique_units,1) # calculating the number of times each distinct value appears
  avpu
}

### applying the function to each of the columns
apply(train, MARGIN = 2, avpu_func)
```

```
##      userid      bookid      rating      age      country
##      4.1        2.2      13624.7     491.7      339.4
##      author      year      publisher  userid_author
##      3.0        678.2        12.5        1.1
```

```
### examining the distribution of the average rating per user
core %>%
  group_by(userid) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```

```
## Error in group_by(., userid): object 'core' not found
```

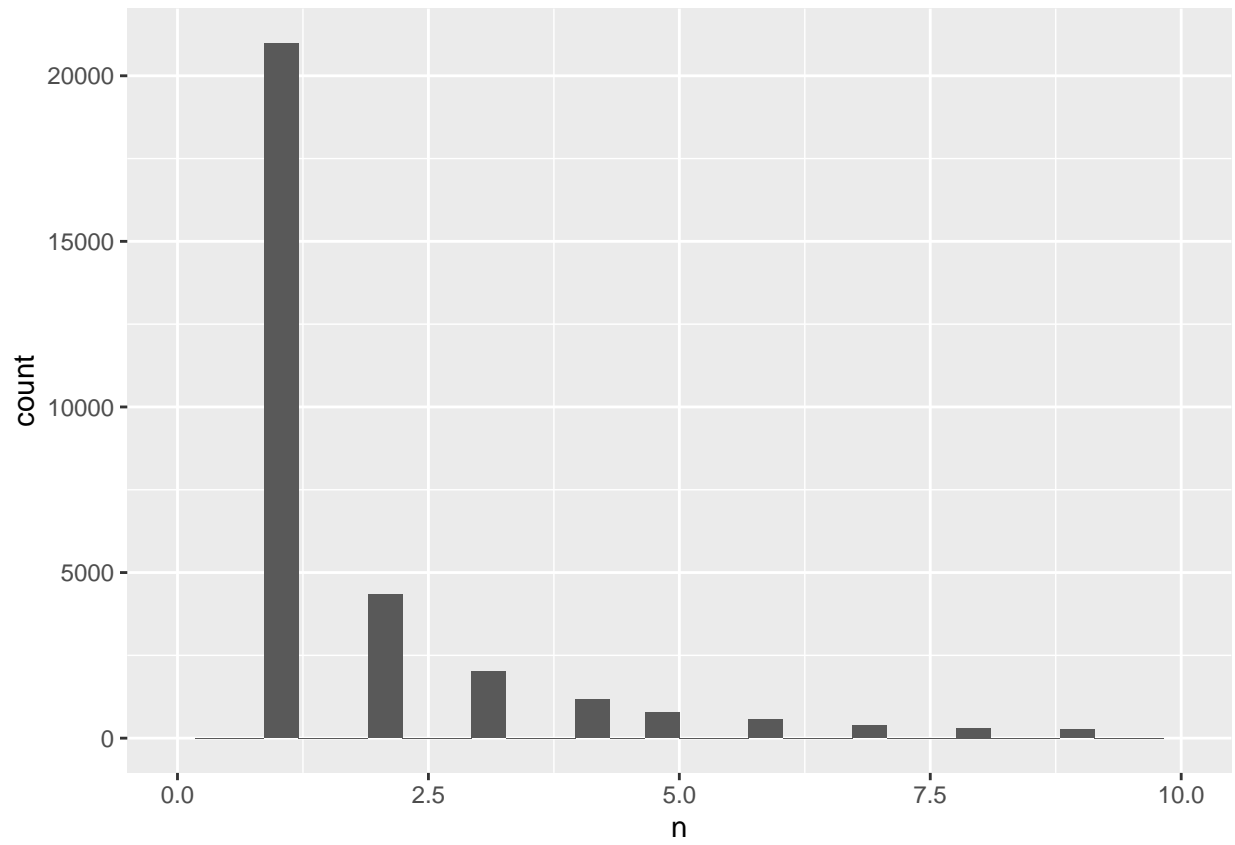
```
### examining the distribution of the number of ratings per user
ratings_per_user<-train %>%
  filter(!is.na(rating) & !is.na(userid)) %>%
  count(userid)

ratings_per_user %>% ggplot(aes(x=n)) +
  geom_histogram() +
  xlim(0,10)
```

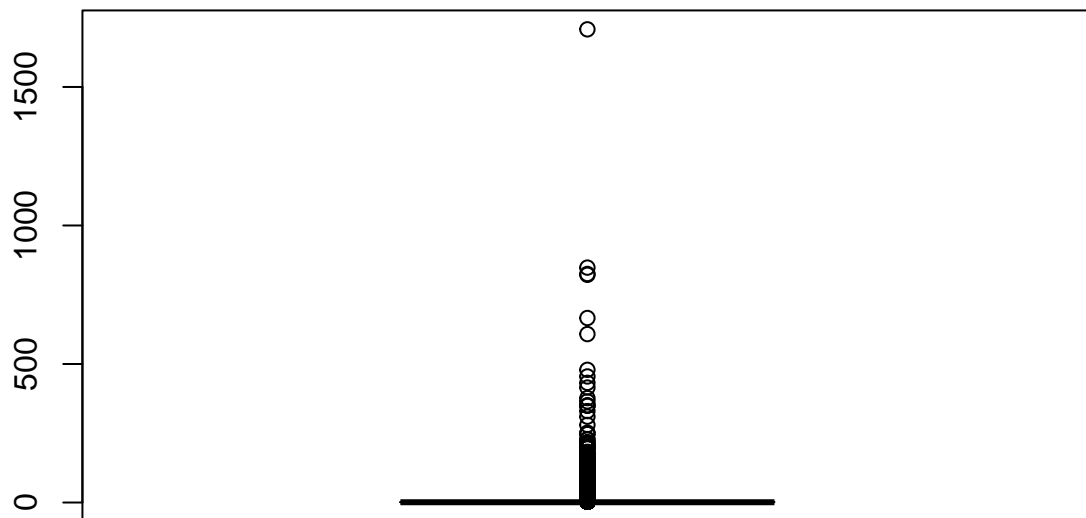
```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```

```
## Warning: Removed 2078 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



```
boxplot(ratings_per_user$n)
```



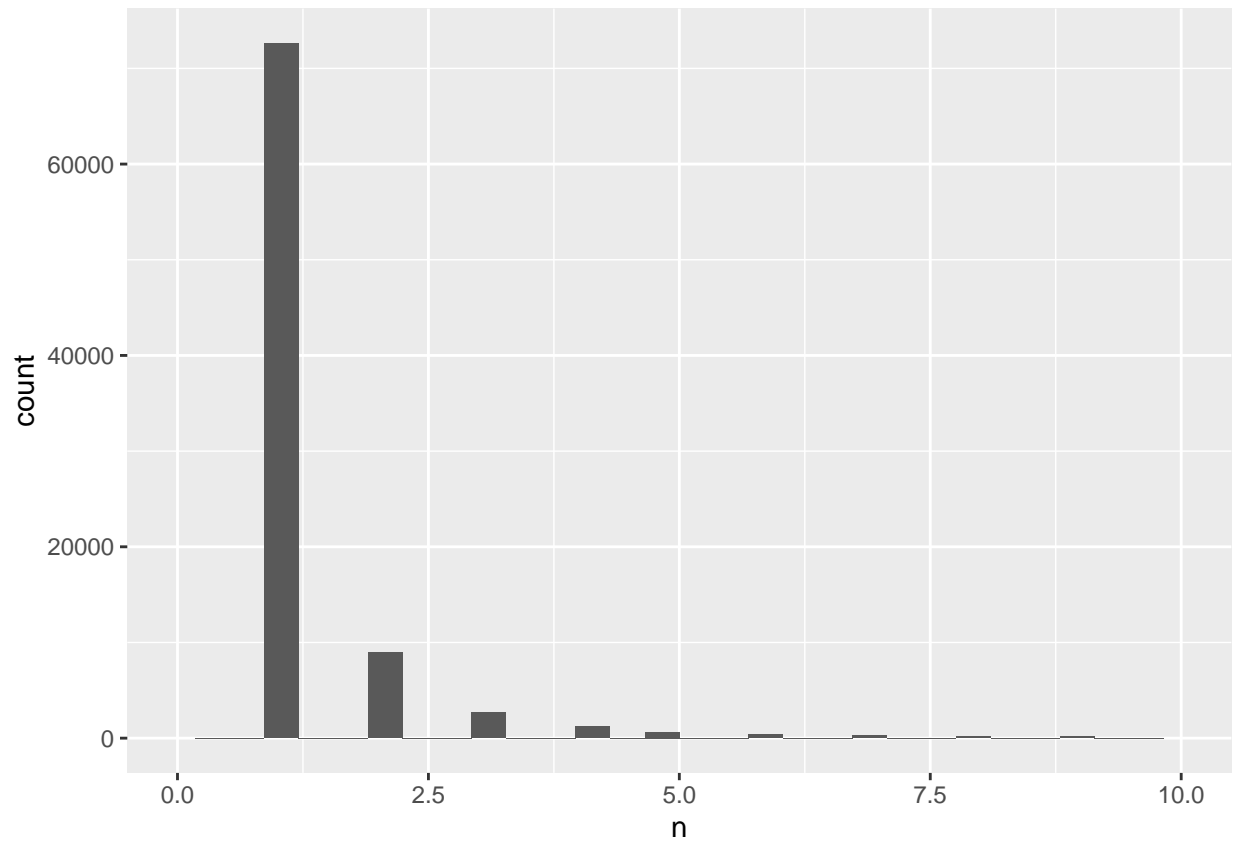
```
### examining the distribution of the number of ratings per book
ratings_per_book<-train %>%
  filter(!is.na(rating) & !is.na(bookid)) %>%
  count(bookid)

ratings_per_book %>% ggplot(aes(x=n)) +
  geom_histogram() +
  xlim(0,10)
```

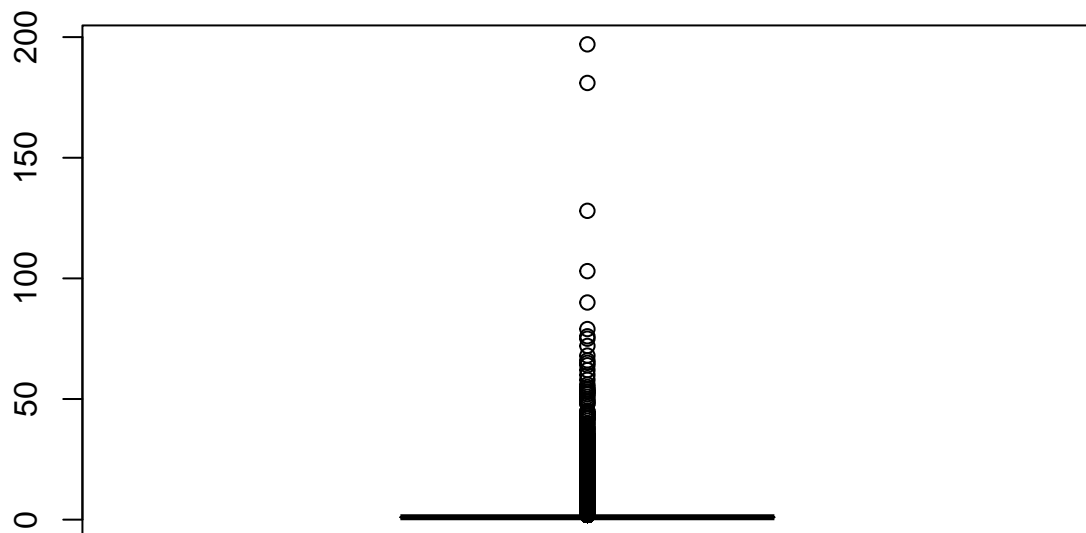
```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```

```
## Warning: Removed 640 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



```
boxplot(ratings_per_book$n)
```



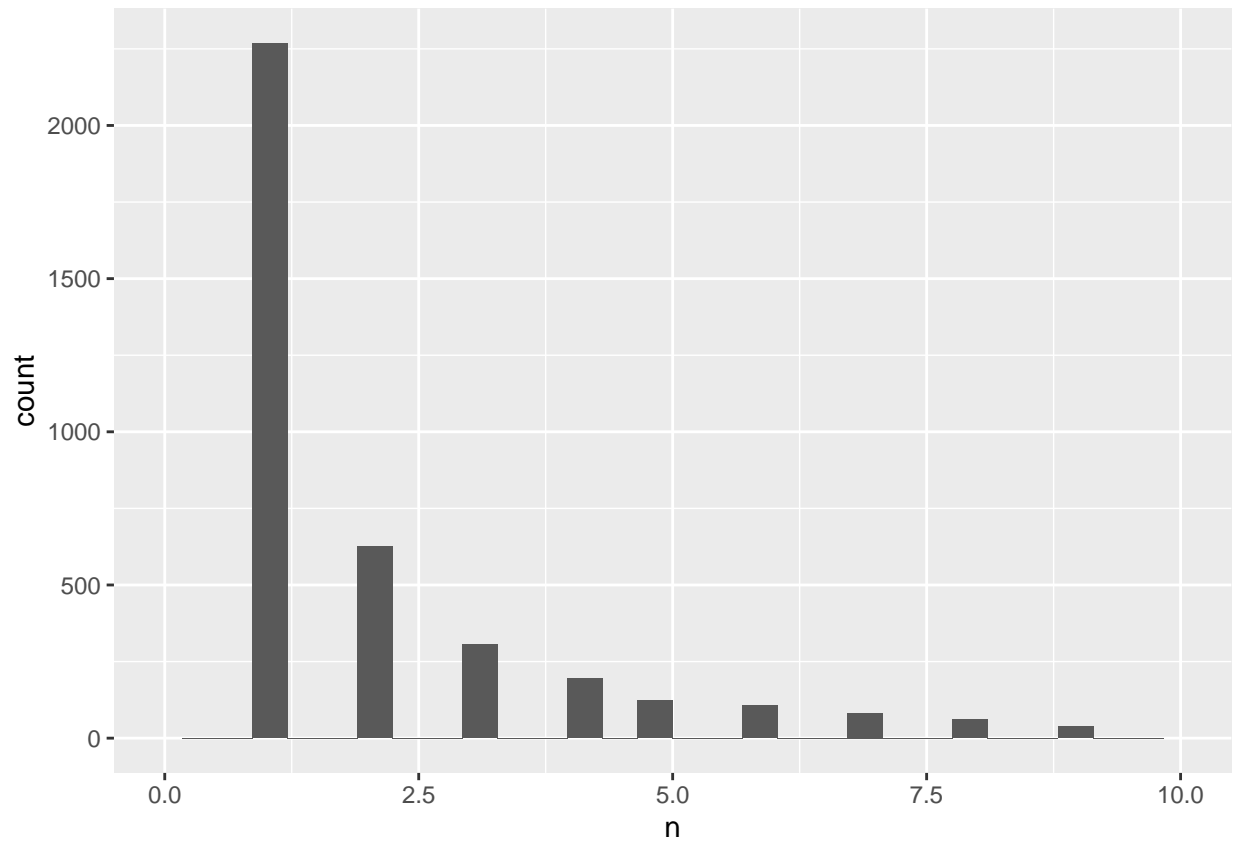
```
### examining the distribution of the number of ratings per publisher
ratings_per_publisher<-train %>%
  filter(!is.na(rating) & !is.na(publisher)) %>%
  count(publisher)

ratings_per_publisher %>% ggplot(aes(x=n)) +
  geom_histogram() +
  xlim(0,10)
```

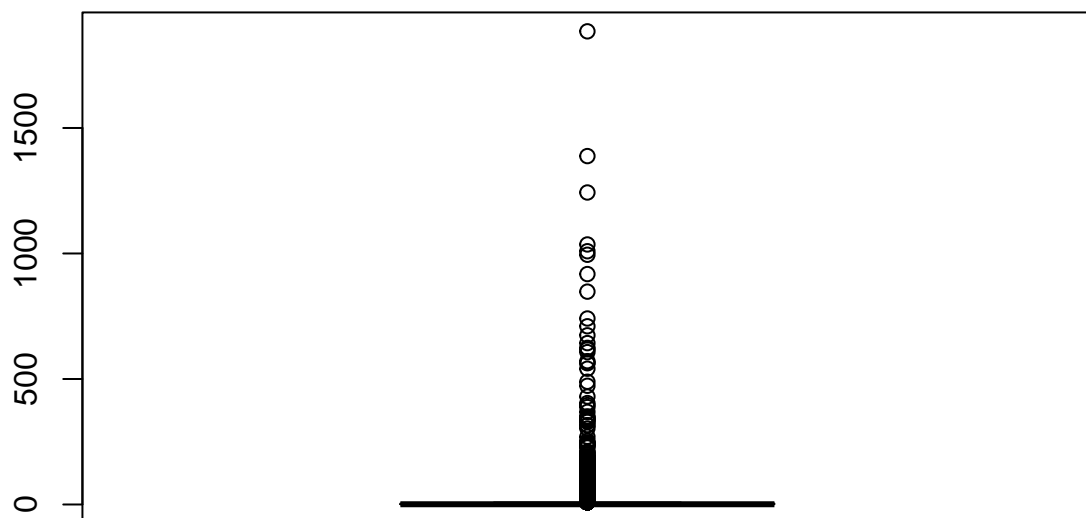
```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```

```
## Warning: Removed 548 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```

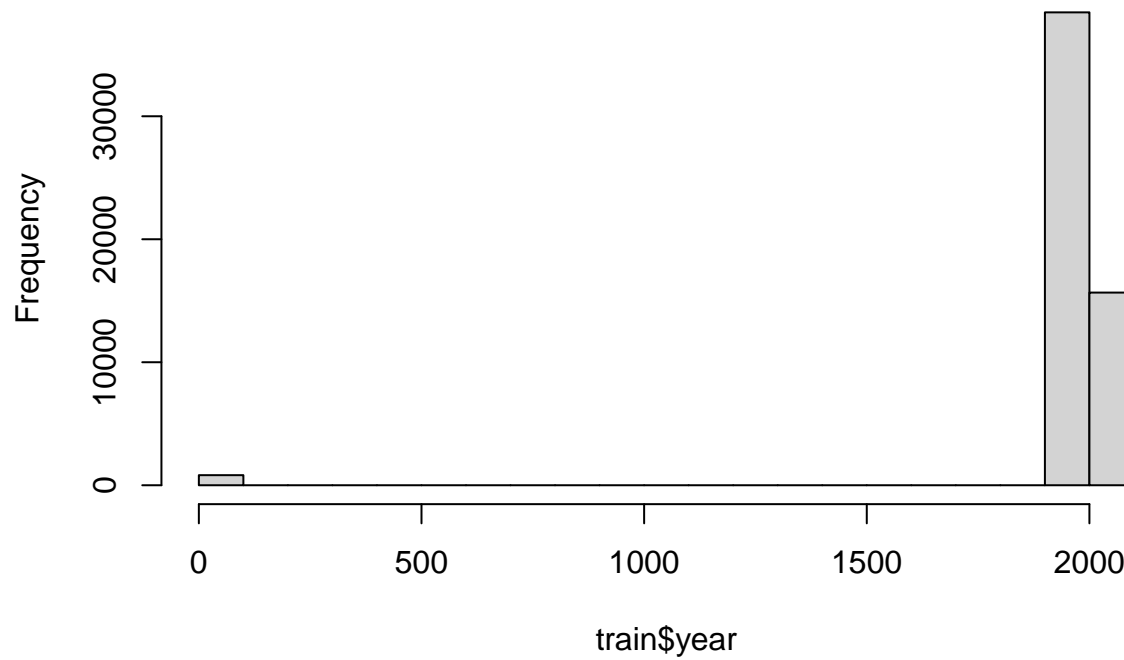


```
boxplot(ratings_per_publisher$n)
```

```
### examining the distribution of the number of ratings per year  
hist(train$year)
```

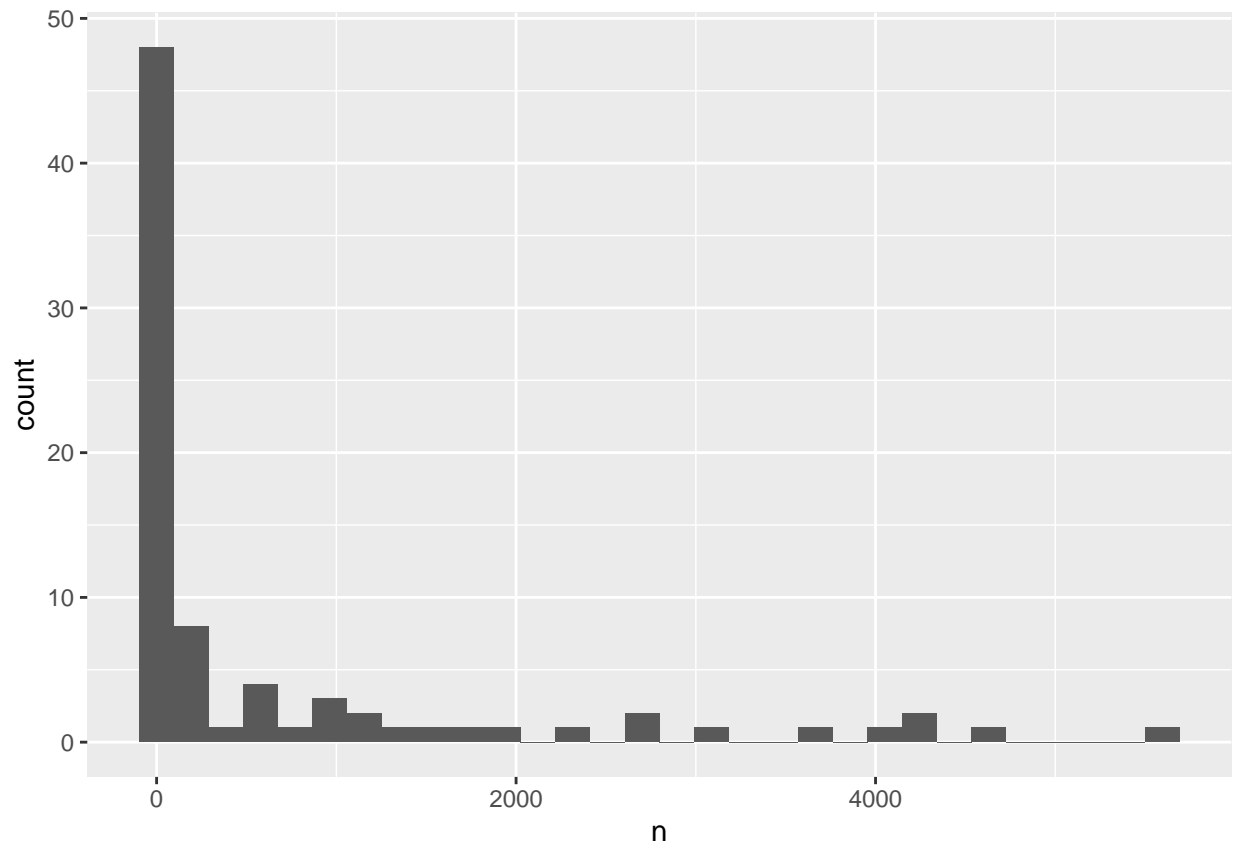
Histogram of train\$year



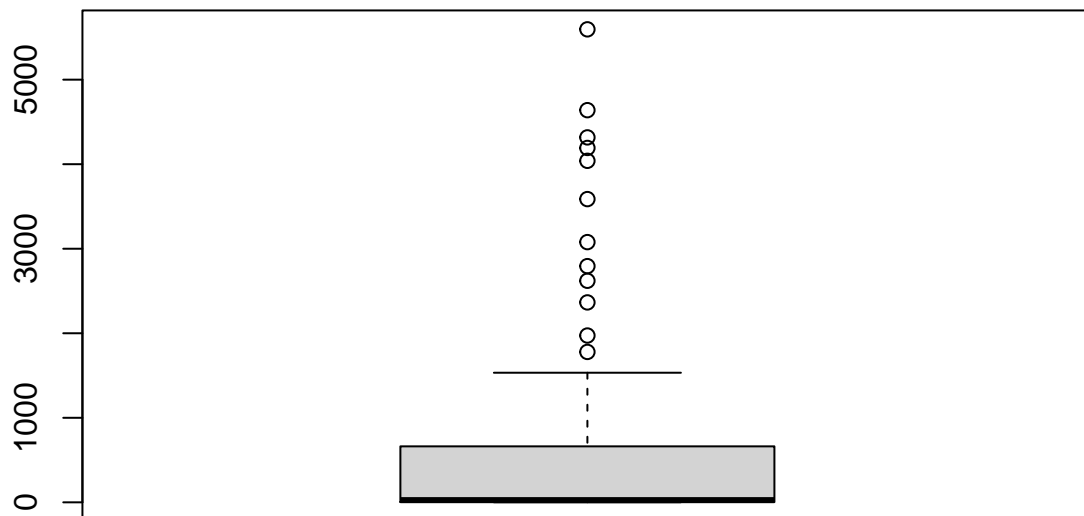
```
ratings_per_year<-train %>%  
  filter(!is.na(rating) & !is.na(year)) %>%  
  count(year)
```

```
ratings_per_year %>% ggplot(aes(x=n)) +  
  geom_histogram()
```

```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```



```
boxplot(ratings_per_year$n)
```



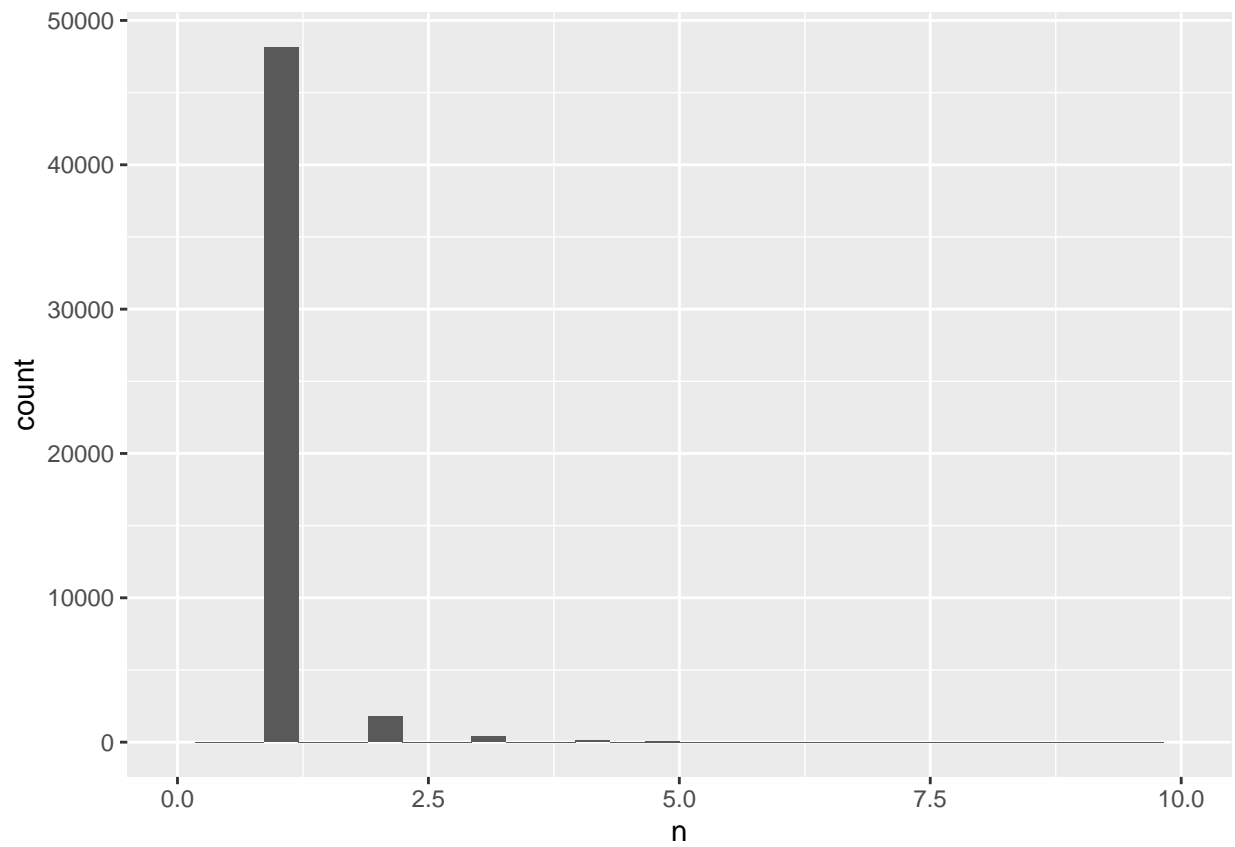
```
### examining the distribution of the number of ratings per user_author pair
ratings_per_userid_author<-train %>%
  filter(!is.na(rating) & !is.na(userid_author)) %>%
  count(userid_author)

ratings_per_userid_author %>% ggplot(aes(x=n)) +
  geom_histogram() +
  xlim(0,10)
```

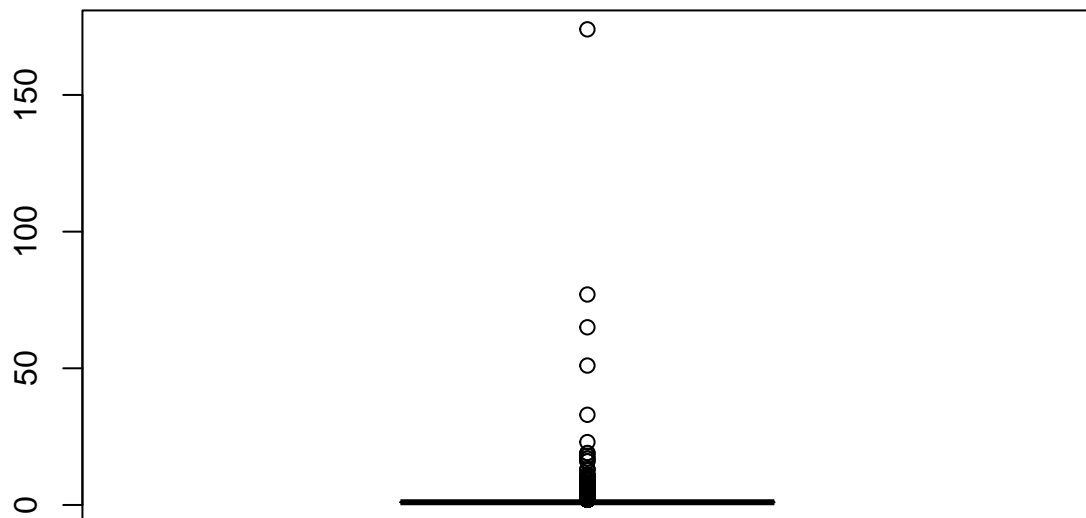
'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.

Warning: Removed 21 rows containing non-finite values (stat_bin).

Warning: Removed 2 rows containing missing values (geom_bar).



```
boxplot(ratings_per_userid_author$n)
```



```
### examining the distribution of age
```

```
age_without_nas<-train$age[!is.na(train$age) & train$age!="NULL"]
length(age_without_nas)
```

```
## [1] 52607
```

```
length(train$age)
```

```
## [1] 136247
```

```
n_distinct(age_without_nas)
```

```
## [1] 107
```

```
n_distinct(train$age)
```

```
## [1] 108
```

```
head(age_without_nas)
```

```
## [1] 16 16 25 25 25 19
```

```
class(age_without_nas)
```

```
## [1] "numeric"
```

```
length(age_without_nas)
```

```
## [1] 52607
```

```
hist(age_without_nas)
```

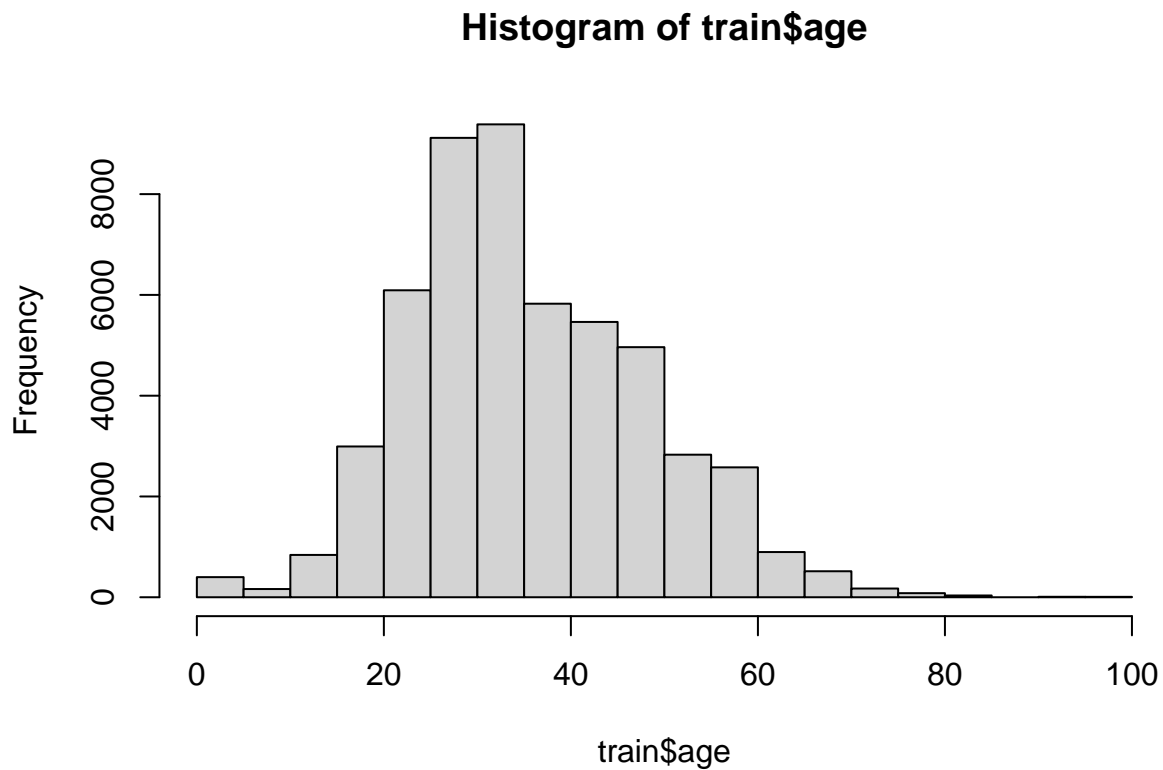


```
sum(age_without_nas>100, na.rm=T)
```

```
## [1] 240
```

```
### removing rows with age>100 since that is probably false  
train$age[train$age>100]<-NA
```

```
### examining the age distribution  
hist(train$age)
```

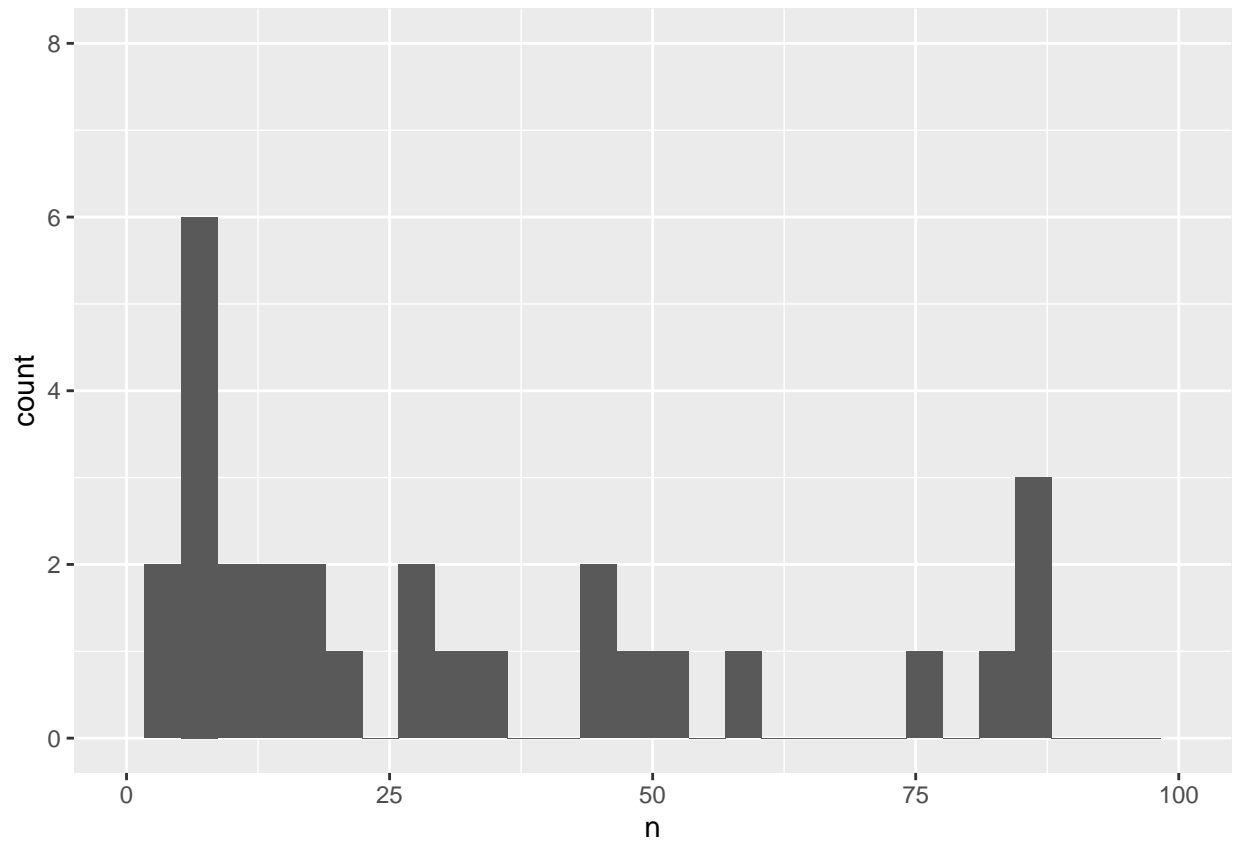


```
ratings_per_age<-train %>%  
  filter(!is.na(rating) & !is.na(age) & train$age!="NULL") %>%  
  count(age)  
  
ratings_per_age %>% ggplot(aes(x=n)) +  
  geom_histogram() +  
  xlim(0,100)
```

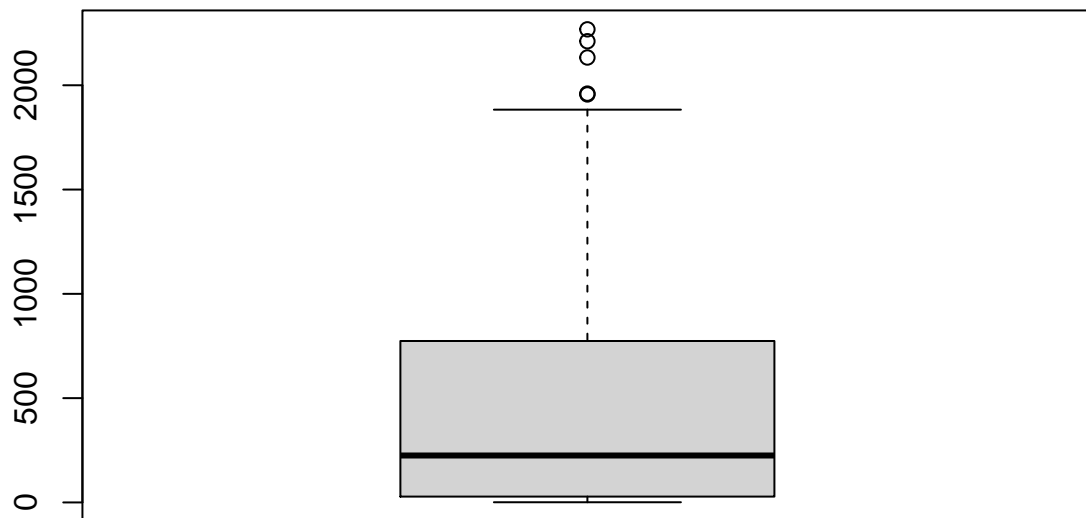
```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```

```
## Warning: Removed 56 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```

```
boxplot(ratings_per_age$n)
```



```
### examining the distribution of the age brackets
table(train$age_bracket)
```

```
## < table of extent 0 >
```

```
train %>% ggplot(aes(x=age_bracket)) +
  geom_bar()
```

```
## Error in FUN(X[[i]], ...): object 'age_bracket' not found
```

```

### saving the train set
saveRDS(train, "train")

### cleaning the working space
rm(list=ls())

### cleaning memory
invisible(gc())

### reloading the train set
train<-readRDS("train")

```

4. Evaluating simple models

```

#####
### 4. A simple model ###
#####

### To keep things simple, we will use the method of "Leave One Out Cross Validation" (LOOCV)

### creating a subset of the training set, for evaluating the model ###
### creating index of 20%
suppressWarnings(set.seed(1, sample.kind="Rounding")) # setting the seed
test_index <- createDataPartition(y = train$rating, times = 1, p = 0.2, list = FALSE) # defining a 50%

```

```
core <- train[-test_index,]  
temp <- train[test_index,]
```

```
# making sure that the temporary split is 80-20  
length(test_index)/nrow(train)
```

```
## [1] 0.2000117
```

```
# creating a 'sub' set that includes only userids and bookids that are also in the training set  
sub <- temp %>%  
  semi_join(train, by = "bookid") %>%  
  semi_join(train, by = "userid")
```

```
# adding rows that were included in the index, but not in the test set, to the training set  
removed <- anti_join(temp, sub)
```

```
## Joining, by = c("userid", "bookid", "rating", "age", "country", "author", "year", "publisher", "user")
```

```
core <- rbind(core, removed)
```

```
### making sure that the number of rows in the training and test sets is equal to  
### the number of rows in the full dataset  
nrow(train)-(nrow(core)+nrow(sub)) # this should be zero
```

```
## [1] 0
```

```
### checking the final relative sizes of the training and test sets  
nrow(core)/nrow(train) # the core set is ~80% of the training data
```

```
## [1] 0.7999883
```

```
nrow(sub)/nrow(train) # the sub set is ~20% of the training data
```

```
## [1] 0.2000117
```

```
### The first model ###  
# Predicting only according to the average rating in the core dataset ###  
mu <- mean(core$rating)  
mu
```

```
## [1] 7.519716
```

```
# checking the root mean squared error  
core$mean<-mu  
naive_rmse <- RMSE(core$rating, core$mean)  
naive_rmse
```

```
## [1] 1.861179
```

```
# creating a results table
rmse_results <- c("Just the average", round(naive_rmse,5))
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

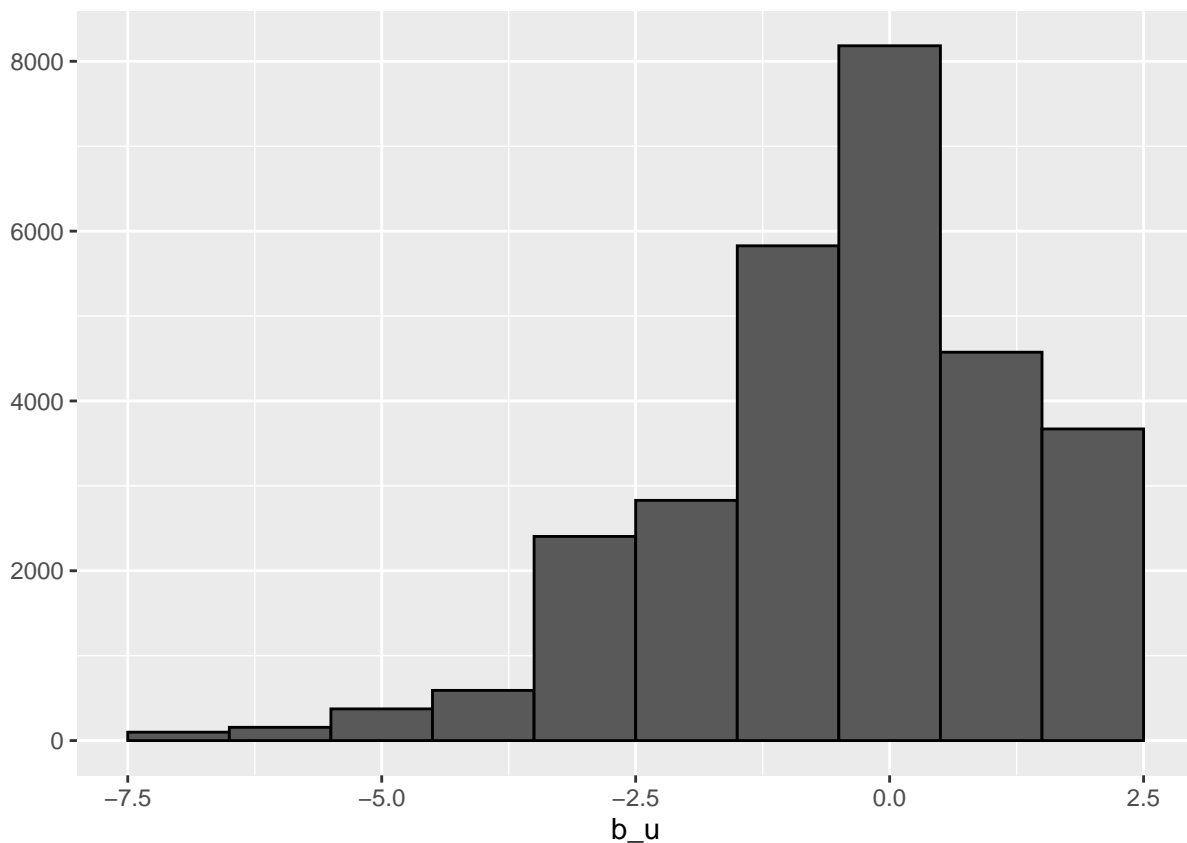
```
##           method           RMSE
## "Just the average"      "1.86118"
```

```
### adding user effects ###
```

```
user_avgs <- core %>%
  group_by(userid) %>%
  summarize(b_u = mean(rating - mu))
```

```
# examining the distributions of user effects
```

```
qplot(b_u, data = user_avgs, bins = 10, color = I("black"))
```



```
# checking the rmse
```

```
predicted_ratings <- mu + sub %>%
  left_join(user_avgs, by='userid') %>%
  pull(b_u)
user_effects_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)
```

```
rmse_results <- rbind.data.frame(rmse_results, c("User effects", round(user_effects_rmse,5)))
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##           method    RMSE
## 1 Just the average 1.86118
## 2      User effects 1.69647
```

```
### adding book effects ###
# estimating the book effects, by computing
# the overall average and the user effect, and then
# the book effect is the average of the remainder, after they
# are subtracted from the rating (book effect= average of (rating - overall average - user effect))
age_bracket_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
  group_by(age_bracket) %>%
  summarize(b_a = mean(rating - mu - b_u))
```

```
## Error: Must group by variables found in '.data'.
## * Column 'age_bracket' is not found.
```

```
# examining the distributions of age_bracket effects
qplot(b_a, data = age_bracket_avgs, bins = 10, color = I("black"))
```

```
## Error in eval_tidy(mapping$x, data, caller_env): object 'age_bracket_avgs' not found
```

```
# checking the rmse
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(age_bracket_avgs, by='age_bracket') %>%
  mutate(pred = mu + b_u + b_a) %>%
  pull(pred)
```

```
## Error in is.data.frame(y): object 'age_bracket_avgs' not found
```

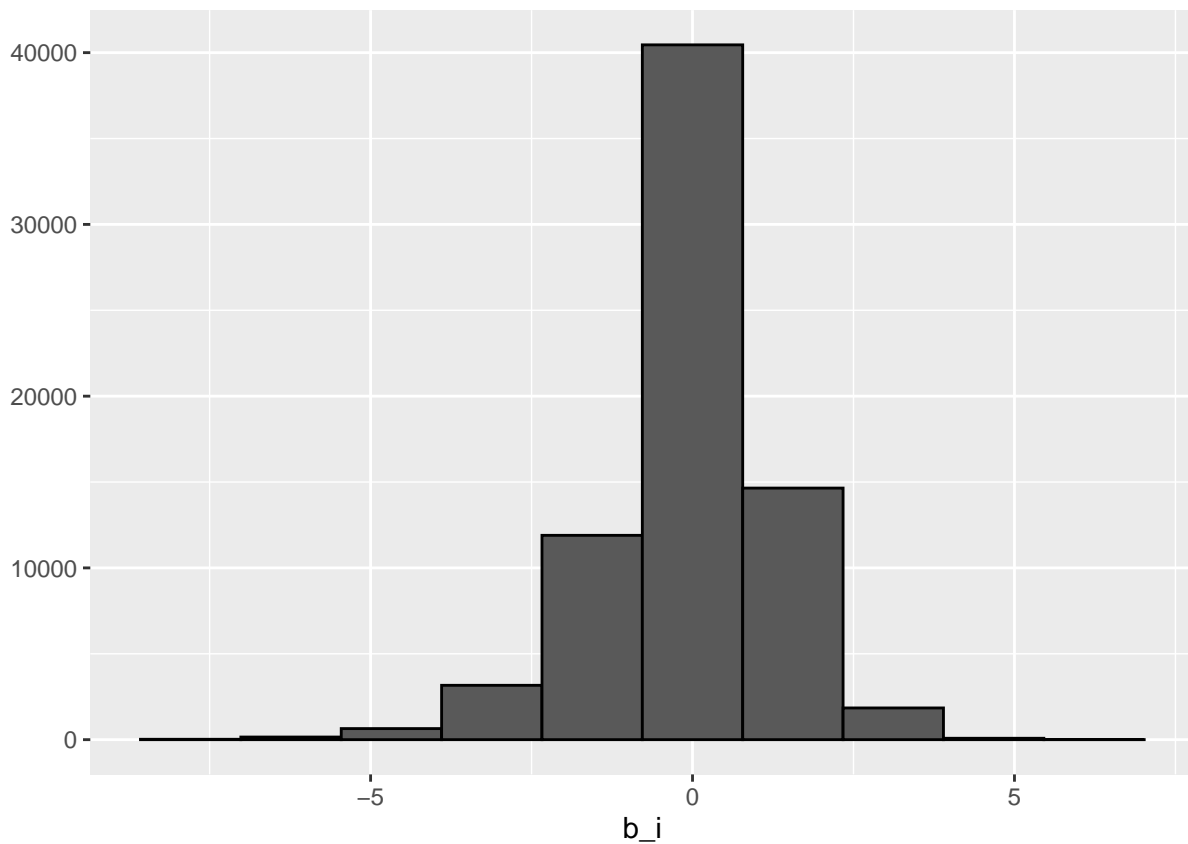
```
user_and_age_bracket_effects_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)

rmse_results <- rbind.data.frame(rmse_results, c("User and age_bracket effects", round(user_and_age_bracket_effects_rmse, 4)))
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##           method    RMSE
## 1      Just the average 1.86118
## 2      User effects 1.69647
## 3 User and age_bracket effects 1.69647
```

```
### examining book effects ###
# estimating the book effects, by computing
# the overall average and the user effect, and then
# the book effect is the average of the remainder, after they
# are subtracted from the rating (book effect= average of (rating - overall average - user effect))
book_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
  group_by(bookid) %>%
  summarize(b_i = mean(rating - mu - b_u))
```

```
# examining the distributions of book effects
qplot(b_i, data = book_avgs, bins = 10, color = I("black"))
```



```
# checking the rmse
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(book_avgs, by='bookid') %>%
  mutate(pred = mu + b_u + b_i) %>%
  pull(pred)
user_and_book_effects_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)

rmse_results <- rbind.data.frame(rmse_results, c("User and book effects", round(user_and_book_effects_rmse, 4)))
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##               method    RMSE
## 1      Just the average 1.86118
## 2           User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4      User and book effects 1.87867
```

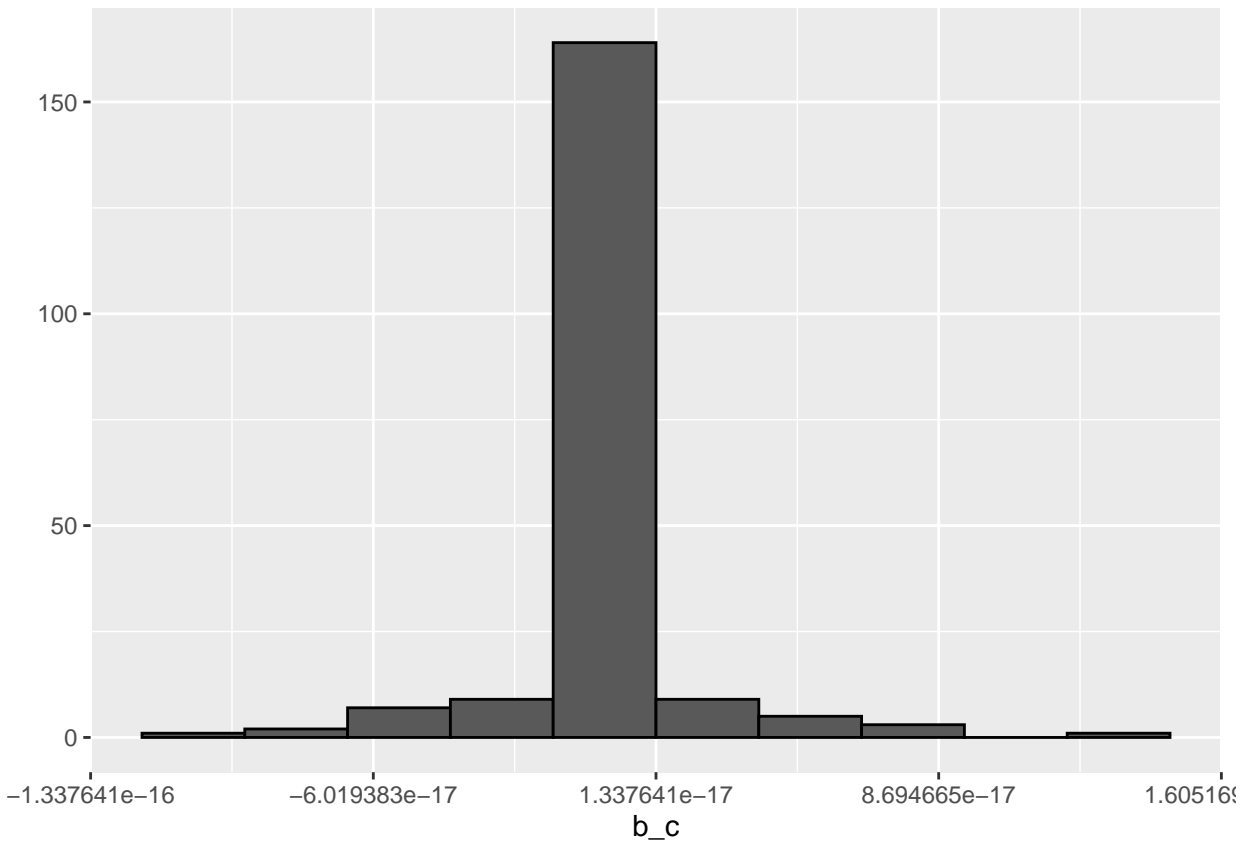
```
### adding country effects ###
country_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
```

```

group_by(country) %>%
  summarize(b_c = mean(rating - mu - b_u))

# examining the distributions of country effects
qplot(b_c, data = country_avgs, bins = 10, color = I("black"))

```



```

# checking the rmse
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(country_avgs, by='country') %>%
  mutate(pred = mu + b_u + b_c) %>%
  pull(pred)
user_and_country_effects_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)

rmse_results <- rbind.data.frame(rmse_results, c("User and country effects", round(user_and_country_eff
names(rmse_results)<-c("method", "RMSE")
rmse_results

```

```

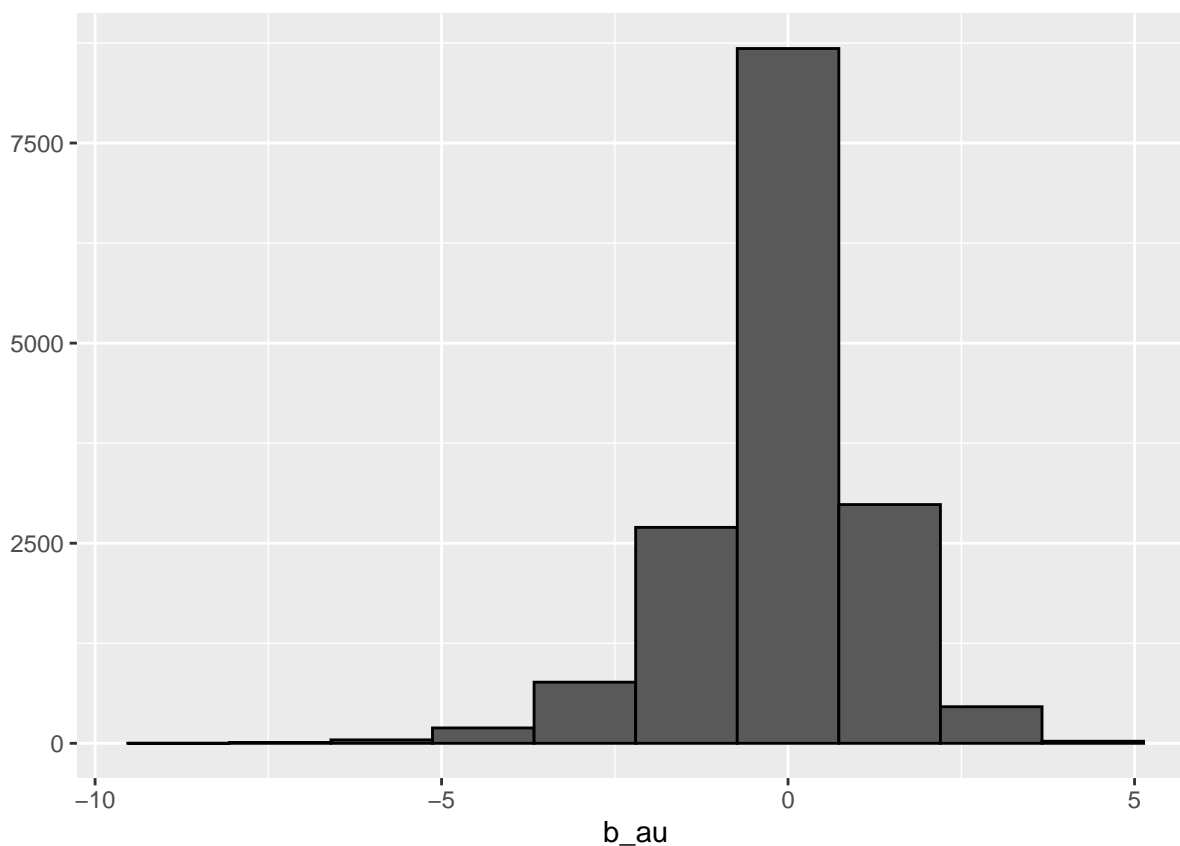
##           method    RMSE
## 1      Just the average 1.86118
## 2           User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4      User and book effects 1.87867
## 5      User and country effects 1.69647

```



```
### adding author effects ###
author_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
  group_by(author) %>%
  summarize(b_au = mean(rating - mu - b_u))

# examining the distributions of author effects
qplot(b_au, data = author_avgs, bins = 10, color = I("black"))
```



```
# checking the rmse
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(author_avgs, by='author') %>%
  mutate(pred = mu + b_u + b_au) %>%
  pull(pred)
user_and_author_effects_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)

rmse_results <- rbind.data.frame(rmse_results, c("User and author effects", round(user_and_author_effects_rmse, 4)))
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##           method    RMSE
## 1      Just the average 1.86118
## 2           User effects 1.69647
## 3 User and age_bracket effects 1.69647
```

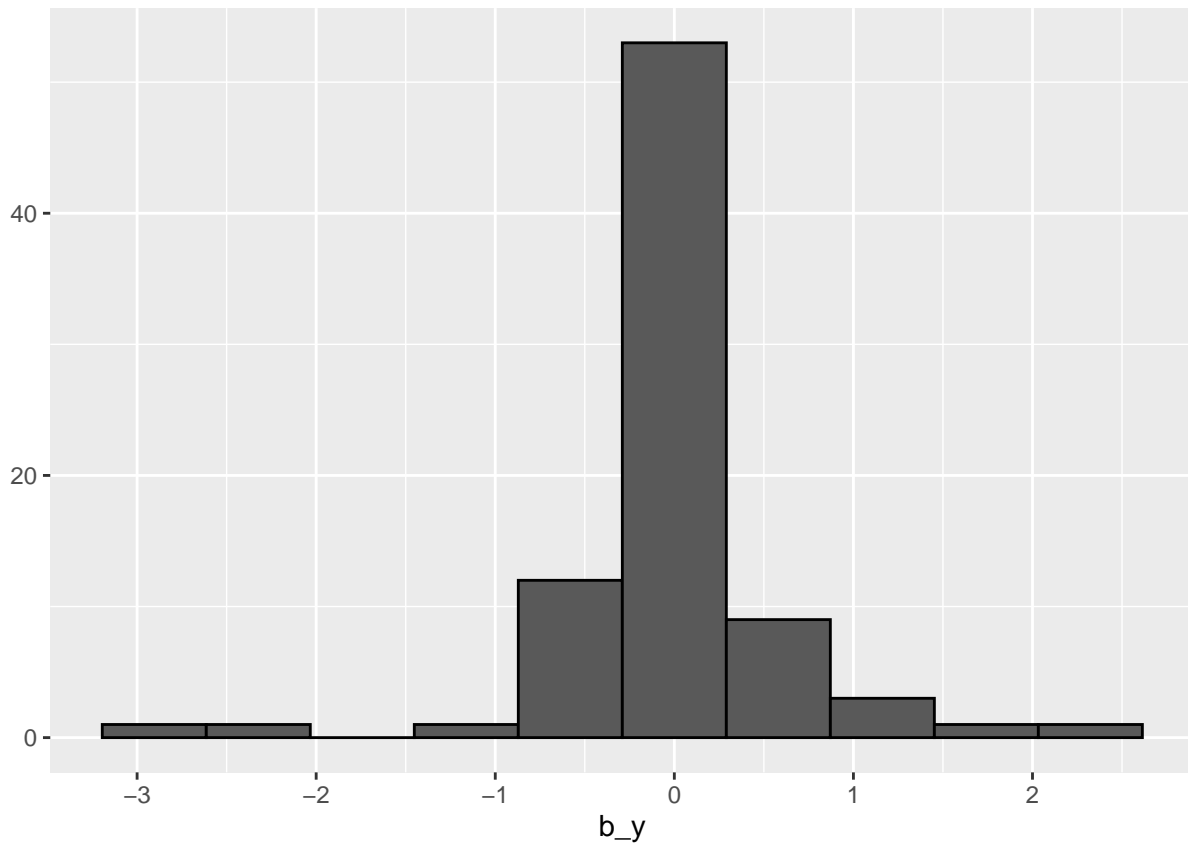
```
## 4      User and book effects 1.87867
## 5      User and country effects 1.69647
## 6      User and author effects 1.71147
```

```
### adding year effects ###
```

```
year_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_u))
```

```
# examining the distributions of year effects
```

```
qplot(b_y, data = year_avgs, bins = 10, color = I("black"))
```



```
# checking the rmse
```

```
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(year_avgs, by='year') %>%
  mutate(pred = mu + b_u + b_y) %>%
  pull(pred)
user_and_year_effects_rmse <- RMSE(predicted_ratings, sub$rating, na.rm = T)
```

```
rmse_results <- rbind.data.frame(rmse_results, c("User and year effects", round(user_and_year_effects_rmse, 4)))
names(rmse_results) <- c("method", "RMSE")
rmse_results
```

```
##              method      RMSE
```

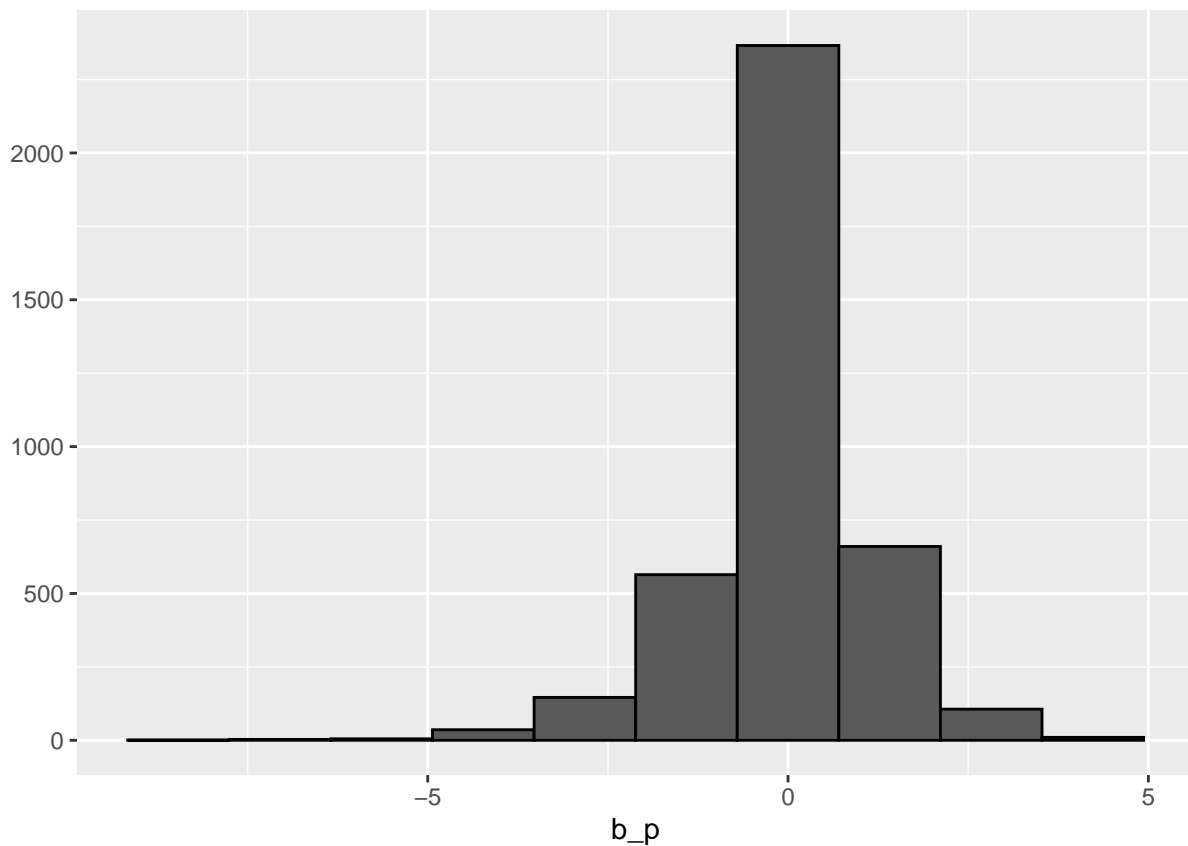
```
## 1          Just the average 1.86118
## 2          User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4          User and book effects 1.87867
## 5          User and country effects 1.69647
## 6          User and author effects 1.71147
## 7          User and year effects 1.69625
```

```
### adding publisher effects ###
```

```
publisher_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
  group_by(publisher) %>%
  summarize(b_p = mean(rating - mu - b_u))
```

```
# examining the distributions of publisher effects
```

```
qplot(b_p, data = publisher_avgs, bins = 10, color = I("black"))
```



```
# checking the rmse
```

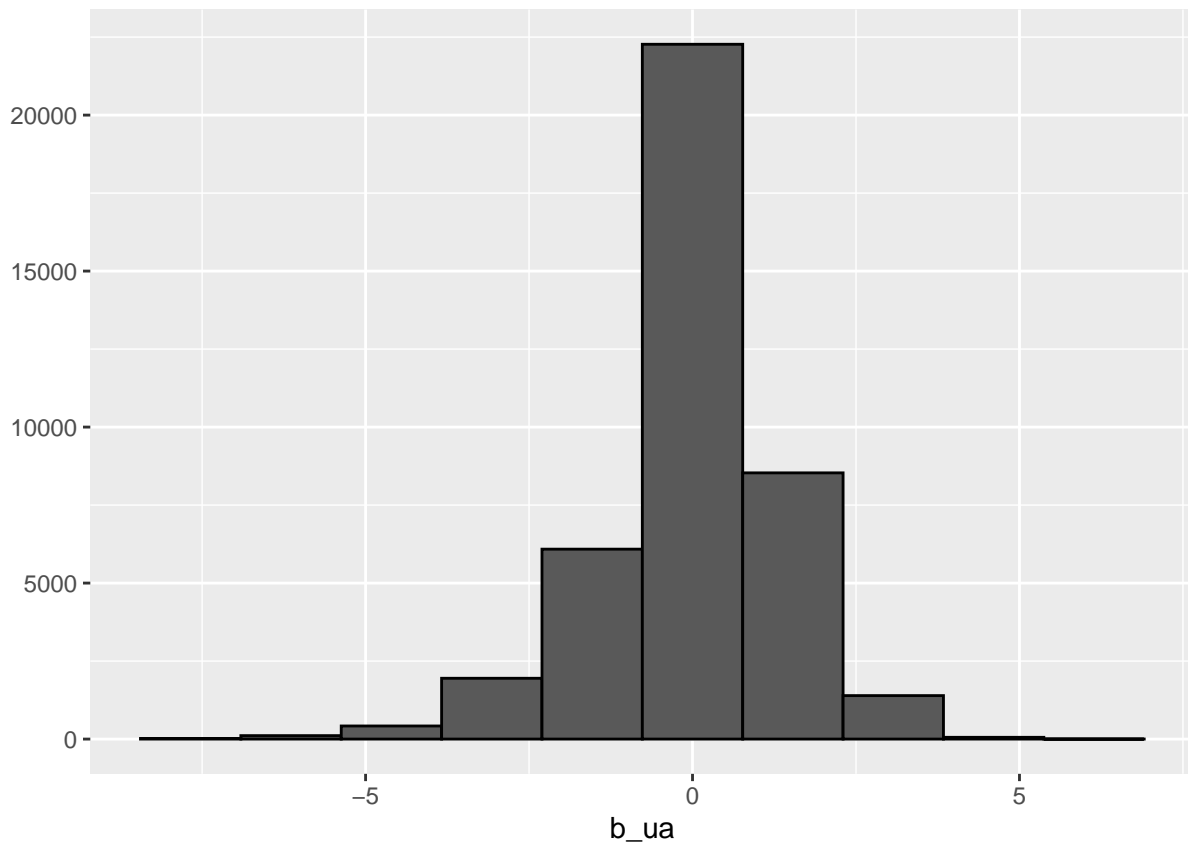
```
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(publisher_avgs, by='publisher') %>%
  mutate(pred = mu + b_u + b_p) %>%
  pull(pred)
user_and_publisher_effects_rmse <- RMSE(predicted_ratings, sub$rating, na.rm = T)
```

```
rmse_results <- rbind.data.frame(rmse_results, c("User and publisher effects", round(user_and_publisher_
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##               method    RMSE
## 1      Just the average 1.86118
## 2           User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4      User and book effects 1.87867
## 5      User and country effects 1.69647
## 6      User and author effects 1.71147
## 7      User and year effects 1.69625
## 8      User and publisher effects 1.69914
```

```
### adding userid_author effects ###
userid_author_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
  group_by(userid_author) %>%
  summarize(b_ua = mean(rating - mu - b_u))
```

```
# examining the distributions of userid_author effects
qplot(b_ua, data = userid_author_avgs, bins = 10, color = I("black"))
```



```

# checking the rmse
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(userid_author_avgs, by='userid_author') %>%
  mutate(pred = mu + b_u + b_ua) %>%
  pull(pred)
user_and_userid_author_effects_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)

rmse_results <- rbind.data.frame(rmse_results, c("User and User*Author effects", round(user_and_userid_
names(rmse_results)<-c("method", "RMSE")
rmse_results

```

```

##              method    RMSE
## 1      Just the average 1.86118
## 2      User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4      User and book effects 1.87867
## 5      User and country effects 1.69647
## 6      User and author effects 1.71147
## 7      User and year effects 1.69625
## 8      User and publisher effects 1.69914
## 9 User and User*Author effects 1.64854

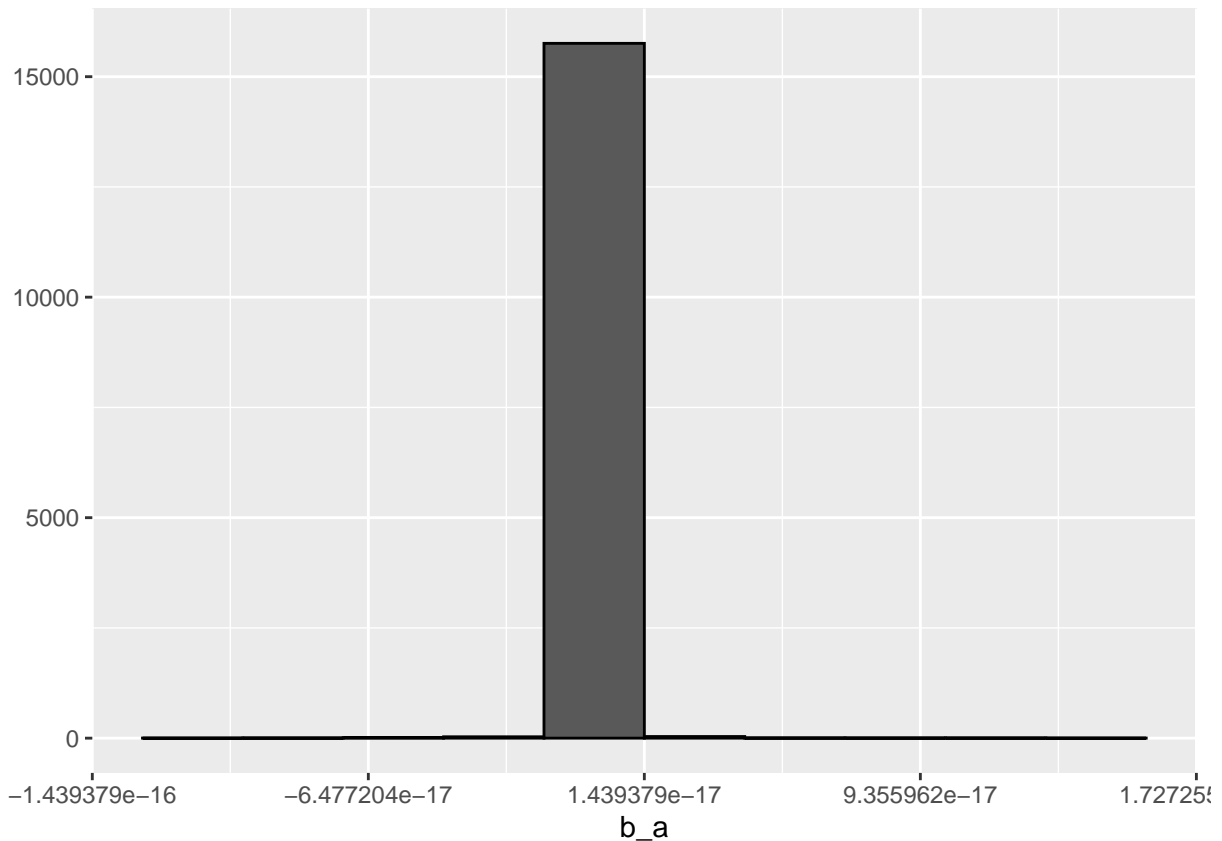
```

```

### adding author effects ###
author_avgs <- core %>%
  left_join(user_avgs, by='userid') %>%
  left_join(userid_author_avgs, by='userid_author') %>%
  group_by(author) %>%
  summarize(b_a = mean(rating - mu - b_u - b_ua))

# examining the distributions of userid_author effects
qplot(b_a, data = author_avgs, bins = 10, color = I("black"))

```



```
# checking the rmse
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
  left_join(userid_author_avgs, by='userid_author') %>%
  left_join(author_avgs, by='author') %>%
  mutate(pred = mu + b_u + b_ua + b_a) %>%
  pull(pred)
user_and_userid_author_and_author_effects_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)

rmse_results <- rbind.data.frame(rmse_results, c("User and userid_author and author effects", round(user_and_userid_author_and_author_effects_rmse, 4)))
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##               method    RMSE
## 1           Just the average 1.86118
## 2             User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4   User and book effects 1.87867
## 5   User and country effects 1.69647
## 6   User and author effects 1.71147
## 7   User and year effects 1.69625
## 8   User and publisher effects 1.69914
## 9   User and User*Author effects 1.64854
## 10 User and userid_author and author effects 1.64854
```

```
names(train)
```

```
## [1] "userid"      "bookid"      "rating"      "age"
## [5] "country"     "author"      "year"        "publisher"
## [9] "userid_author"
```

5. Adding regularization

Adding regularization, to account for the difference between average ratings provided by a few users and average ratings provided by many users.

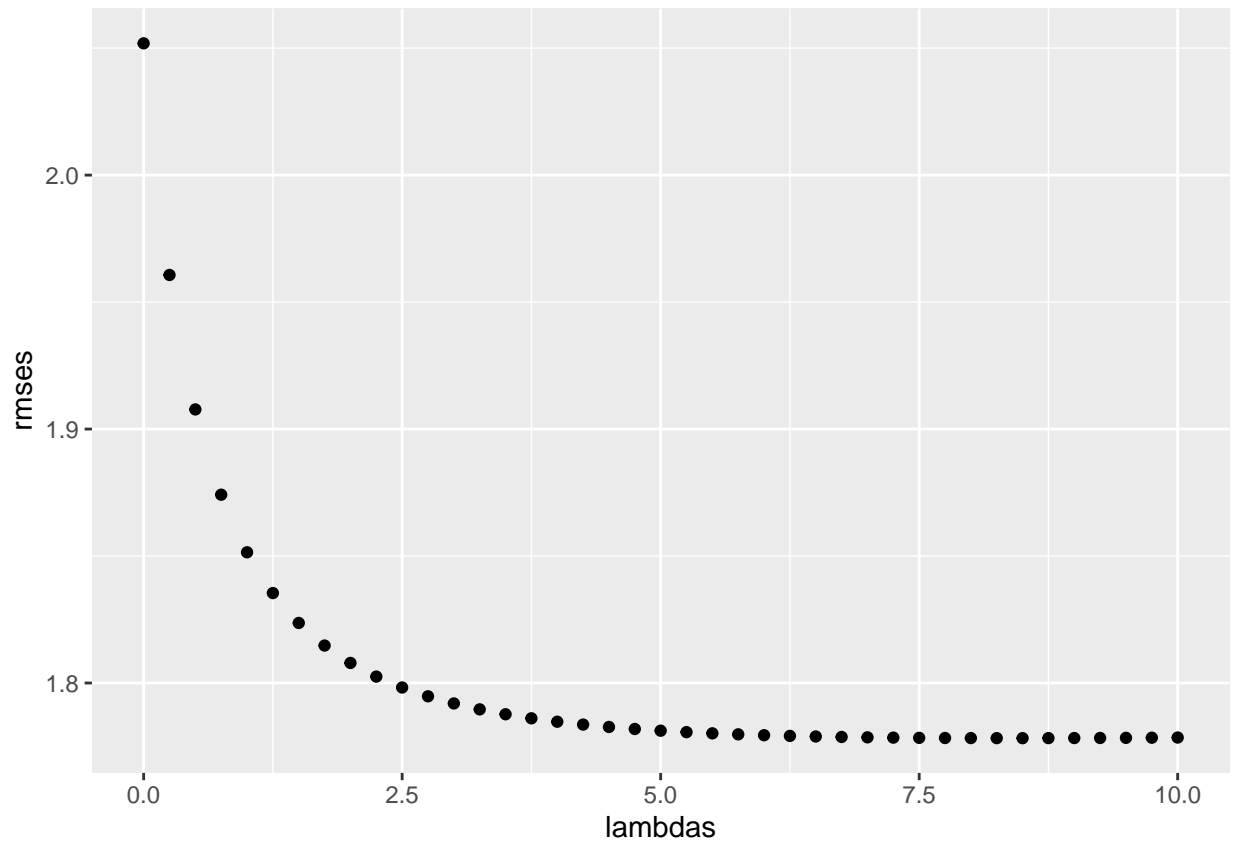
```
#####
### 5. Adding regularization ###
#####

### We will use the model with user and userid_author effects,
### since it produced the lowest RMSE (and adding author effects did not make any difference)

### Choosing lambda
lambdas <- seq(0, 10, 0.25)

mu <- mean(core$rating)
just_the_sum <- core %>%
  group_by(bookid) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmsees <- sapply(lambdas, function(l){
  predicted_ratings <- sub %>%
    left_join(just_the_sum, by='bookid') %>%
    mutate(b_i = s/(n_i+1)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, sub$rating))
})
qplot(lambdas, rmsees)
```



```
lambdas[which.min(rmses)]
```

```
## [1] 8.5
```

```
### Regularizing with lambda = 3
lambda <- 8.5
mu <- mean(core$rating)
mu
```

```
## [1] 7.519716
```

```
### user effects are calculated in the same way as without regularization
user_avgs <- core %>%
  group_by(userid) %>%
  summarize(b_u = mean(rating - mu))

### userid_author effects are calculated differently
userid_author_avgs_reg <- core %>%
  left_join(user_avgs, by='userid') %>%
  group_by(userid_author) %>%
  summarize(b_u = sum(rating - mu - b_u)/(n()+lambda), n_u = n())

### Calculating RMSE ###
predicted_ratings <- sub %>%
  left_join(user_avgs, by='userid') %>%
```



```

left_join(userid_author_avgs_reg, by='userid_author') %>%
  mutate(pred = mu + b_u + b_ua) %>%
  pull(pred)
user_and_userid_author_effects_with_regularization_rmse<-RMSE(predicted_ratings, sub$rating, na.rm = T)

# checking the RMSE
rmse_results <- rbind.data.frame(rmse_results, c("User and User*Author effects with regularization", rownames(rmse_results)))
names(rmse_results)<-c("method", "RMSE")
rmse_results

```

```

##                                method    RMSE
## 1                        Just the average 1.86118
## 2                        User effects 1.69647
## 3      User and age_bracket effects 1.69647
## 4      User and book effects 1.87867
## 5      User and country effects 1.69647
## 6      User and author effects 1.71147
## 7      User and year effects 1.69625
## 8      User and publisher effects 1.69914
## 9      User and User*Author effects 1.64854
## 10     User and userid_author and author effects 1.64854
## 11 User and User*Author effects with regularization 1.6516

```

```

### saving files
saveRDS(core, "core")
saveRDS(sub, "sub")
saveRDS(train, "train")
saveRDS(rmse_results, "rmse_results")

```

6. Evaluating other algorithms: Popular, SVD, SVDF

Now let us evaluate more advanced prediction models, namely: 1. 'Popular' 2. Singular Value Decomposition (SVD) 3. Funk Singular Value Decomposition (SVDF)

```

#####
### 6. Other algorithms ###
#####

### cleaning the working space
rm(list=ls())

### cleaning memory
invisible(gc())

### reloading the train set
train<-readRDS("train")
test<-readRDS("test")
rmse_results<-readRDS("rmse_results")

### Preparing the data ###
### removing books in the training set that do not appear in the test set
trainmat_final <- train %>%

```

```

    semi_join(test, by = "bookid")

### exploring the datasets
dim(train)

## [1] 136247      9

dim(trainmat_final)

## [1] 51498      9

### saving
saveRDS(trainmat_final, file="trainmat_final")

### converting the training set into a matrix
users_and_ratings_train_set<-cbind.data.frame(trainmat_final$userid, trainmat_final$bookid, trainmat_final$rating)
names(users_and_ratings_train_set)<-c("userid", "bookid", "rating")

### exploring
dim(users_and_ratings_train_set)

## [1] 51498      3

head(users_and_ratings_train_set)

##   userid   bookid rating
## 1 276729      NA      3
## 2 276744      NA      7
## 3 276747 679776818      8
## 4 276754 684867621      8
## 5 276755 451166892      5
## 6 276762 380711524      5

### counting missing values
n_missing<-sum(is.na(users_and_ratings_train_set))

### removing rows with missing bookids
### (that is the only column that contains missing values)
index<-which(is.na(users_and_ratings_train_set$bookid))
head(index)

## [1]  1  2  9 15 18 32

length(index)

## [1] 8359

```

```
users_and_ratings_train_set<-users_and_ratings_train_set[-index,]
```

```
### making sure that it worked  
dim(users_and_ratings_train_set)
```

```
## [1] 43139      3
```

```
### making sure that there are no missing values left  
sum(is.na(users_and_ratings_train_set)) # this should be zero
```

```
## [1] 0
```

```
### converting the matrix into a "realRatingMatrix")  
trainmat_final <- as(users_and_ratings_train_set, "realRatingMatrix")  
dim(trainmat_final)
```

```
## [1] 17652 13443
```

```
### saving  
saveRDS(trainmat_final, file="trainmat_final")  
  
### creating a regular matrix  
trainmat_final_reg<-as(trainmat_final, "matrix")  
  
### saving  
saveRDS(trainmat_final_reg, file="trainmat_final_reg")  
  
### exploring the matrix  
class(trainmat_final_reg)
```

```
## [1] "matrix" "array"
```

```
n_missing<-sum(is.na(trainmat_final_reg)) # counting missing values  
n_missing
```

```
## [1] 237252697
```

```
n_non_missing<-sum(!is.na(trainmat_final_reg)) # counting non-missing values  
n_non_missing
```

```
## [1] 43139
```

```
all<-nrow(trainmat_final_reg)*ncol(trainmat_final_reg) # counting all values  
all
```

```
## [1] 237295836
```

```
p_missing<-n_missing/all # calculating the percentage of missing values
p_missing
```

```
## [1] 0.9998182
```

```
p_non_missing<-1-p_missing
p_non_missing
```

```
## [1] 0.0001817942
```

```
rm(trainmat_final_reg, n_missing, all) # removing the matrix and other unnecessary objects

### Increasing memory size
memory.limit(size = 10^10)
```

```
## [1] 1e+10
```

```
### cleaning memory
invisible(gc())

### checking number of ratings per item
number_of_ratings<-colCounts(trainmat_final)
min(number_of_ratings)
```

```
## [1] 1
```

```
max(number_of_ratings)
```

```
## [1] 197
```

```
### exploring a sample of the matrix
trainmat_final@data[1500:1510, 2001:2009]
```

```
## 11 x 9 sparse Matrix of class "dgCMatrix"
##      312966091 312966210 312966393 312966555 312966806 312966970 312967004
## 166778      .      .      .      .      .      .      .
## 166793      .      .      .      .      .      .      .
## 166795      .      .      .      .      .      .      .
## 166799      .      .      .      .      .      .      .
## 166809      .      .      .      .      .      .      .
## 166812      .      .      .      .      .      .      .
## 166813      .      .      .      .      .      .      .
## 166815      .      .      .      .      .      .      .
## 166824      .      .      .      .      .      .      .
## 166825      .      .      .      .      .      .      .
## 166828      .      .      .      .      .      .      .
##      312968191 312968302
## 166778      .      .
## 166793      .      .
```

```
## 166795      .      .
## 166799      .      .
## 166809      .      .
## 166812      .      .
## 166813      .      .
## 166815      .      .
## 166824      .      .
## 166825      .      .
## 166828      .      .
```

```
### normalizing the values
normalize(trainmat_final, method = "Z-score")
```

```
## 17652 x 13443 rating matrix of class 'realRatingMatrix' with 43139 ratings.
## Normalized using z-score on rows.
```

```
### saving
saveRDS(trainmat_final, file="trainmat_final")

### Setting up the evaluation scheme.
### We will use cross-validation

### calculating the mean of all ratings again, in order to determine
### the value of a "good" rating for the evaluation function
train<-readRDS("train")
mean(train$rating)
```

```
## [1] 7.520239
```

```
rm(train) # removing the dataset from the workspace to save memory

Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 19:32:41 CET"
```

```
scheme <- trainmat_final %>%
  evaluationScheme(method = "cross-validation",
    k=2, # 2-fold cross validation
    given = 1,
    goodRating = 8
  )

Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 19:32:43 CET"
```

```
### saving
saveRDS(scheme, file="scheme")

Sys.time()
```

```
## [1] "2022-01-14 19:32:43 CET"
```

```
##### Evaluating the models #####
```

```
### cleaning the working space
```

```
rm(list=ls())
```

```
### cleaning memory
```

```
invisible(gc())
```

```
### loading the scheme and the rmse_results
```

```
scheme<-readRDS("scheme")
```

```
rmse_results<-readRDS("rmse_results")
```

```
### Popular ###
```

```
### evaluating the "Popular" model
```

```
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 19:32:44 CET"
```

```
result_rating_popular <- evaluate(scheme,  
                                  method = "popular",  
                                  parameter = list(normalize = "Z-score"),  
                                  type = "ratings"  
)
```

```
## popular run fold/sample [model time/prediction time]
```

```
## 1 [0.12sec/31.83sec]
```

```
## 2 [0.09sec/31.44sec]
```

```
### examining the results
```

```
results_pop<-result_rating_popular@results %>%
```

```
  map(function(x) x@cm) %>%
```

```
  unlist() %>%
```

```
  matrix(ncol = 3, byrow = T) %>%
```

```
  as.data.frame() %>%
```

```
  summarise_all(mean) %>%
```

```
  setNames(c("RMSE", "MSE", "MAE"))
```

```
results_pop
```

```
##      RMSE      MSE      MAE
```

```
## 1 4.864844 23.66689 3.212177
```

```
### adding the results to the list
```

```
rmse_results <- rbind.data.frame(rmse_results, c("Popular model", round(results_pop$RMSE,5)))
```

```
rmse_results
```

```
##                                method    RMSE
```

```
## 1                        Just the average 1.86118
```

```
## 2                        User effects 1.69647
```

```
## 3          User and age_bracket effects 1.69647
## 4          User and book effects 1.87867
## 5          User and country effects 1.69647
## 6          User and author effects 1.71147
## 7          User and year effects 1.69625
## 8          User and publisher effects 1.69914
## 9          User and User*Author effects 1.64854
## 10         User and userid_author and author effects 1.64854
## 11 User and User*Author effects with regularization 1.6516
## 12          Popular model 4.86484
```

```
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 19:34:11 CET"
```

```
### saving
```

```
saveRDS(result_rating_popular, file="result_rating_popular")
```

```
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 19:34:11 CET"
```

```
### saving
```

```
### saveRDS(result_rating_ubcf, file="result_rating_ubcf")
```

```
### svd ###
```

```
### evaluating the svd model
```

```
result_rating_svd <- evaluate(scheme,
                             method = "svd",
                             parameter = list(normalize = "Z-score", k = 5),
                             type = "ratings"
)
```

```
## svd run fold/sample [model time/prediction time]
```

```
## 1 [9.6sec/125.18sec]
```

```
## 2 [12.25sec/119.14sec]
```

```
### examining the results
```

```
results_svd<-result_rating_svd@results %>%
  map(function(x) x@cm) %>%
  unlist() %>%
  matrix(ncol = 3, byrow = T) %>%
  as.data.frame() %>%
  summarise_all(mean) %>%
  setNames(c("RMSE", "MSE", "MAE"))
```

```
results_svd
```

```
##          RMSE          MSE          MAE
## 1 4.256153 18.11662 2.84132
```

```
### adding the results to the list
rmse_results <- rbind.data.frame(rmse_results, c("Singular Value Decomposition", round(results_svd$RMSE, 4)))
rmse_results
```

```
##           method      RMSE
## 1      Just the average 1.86118
## 2           User effects 1.69647
## 3      User and age_bracket effects 1.69647
## 4      User and book effects 1.87867
## 5      User and country effects 1.69647
## 6      User and author effects 1.71147
## 7      User and year effects 1.69625
## 8      User and publisher effects 1.69914
## 9      User and User*Author effects 1.64854
## 10     User and userid_author and author effects 1.64854
## 11     User and User*Author effects with regularization 1.6516
## 12           Popular model 4.86484
## 13     Singular Value Decomposition 4.25615
```

```
### saving
saveRDS(result_rating_svd, file="result_rating_svd")

Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 19:39:09 CET"
```

```
### svdf ###
```

```
### evaluating
result_rating_svdf <- evaluate(scheme,
                               method = "svdf",
                               parameter = list(normalize = "Z-score", k = 5),
                               type = "ratings"
)
```

```
## svdf run fold/sample [model time/prediction time]
## 1 [480.65sec/536.7sec]
## 2 [438.87sec/513.03sec]
```

```
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 20:12:31 CET"
```

```
### examining the results
results_svdf<-result_rating_svdf@results %>%
  map(function(x) x@cm) %>%
  unlist() %>%
  matrix(ncol = 3, byrow = T) %>%
  as.data.frame() %>%
  summarise_all(mean) %>%
  setNames(c("RMSE", "MSE", "MAE"))

results_svdf
```



```
##          RMSE          MSE          MAE
## 1 4.384726 19.22768 2.959894
```

```
### adding the results to the list
```

```
rmse_results <- rbind.data.frame(rmse_results, c("Funk Singular Value Decomposition", round(results_svd, 4)),
rmse_results
```

```
##
## 1 Just the average 1.86118
## 2 User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4 User and book effects 1.87867
## 5 User and country effects 1.69647
## 6 User and author effects 1.71147
## 7 User and year effects 1.69625
## 8 User and publisher effects 1.69914
## 9 User and User*Author effects 1.64854
## 10 User and userid_author and author effects 1.64854
## 11 User and User*Author effects with regularization 1.6516
## 12 Popular model 4.86484
## 13 Singular Value Decomposition 4.25615
## 14 Funk Singular Value Decomposition 4.38473
```

```
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-14 20:12:31 CET"
```

```
### saving
```

```
saveRDS(result_rating_svdf, file="result_rating_svdf")
saveRDS(rmse_results, file="rmse_results")
```

Since over 98% percent of the values are missing, I add SVD and SVDF to the models that I evaluate. These model are supposed to deal with sparse matrices well.

Since my processing power is limited, I evaluate the models on only 10% of the users in the training set. Otherwise the evaluation takes too much time. I choose the 10% of the users who gave the most ratings.

7. Applying the chosen model to the test set

```
#####
### 7. Applying the chosen algorithm to the test set ###
#####
```

```
### cleaning the working space
rm(list=ls())
```

```
### cleaning memory
invisible(gc())
```

```
### loading the datasets
```

```
train<-readRDS("train")
test<-readRDS("test")
rmse_results<-readRDS("rmse_results")
```

```
### calculating the mean rating
mu <- mean(train$rating)
mu
```

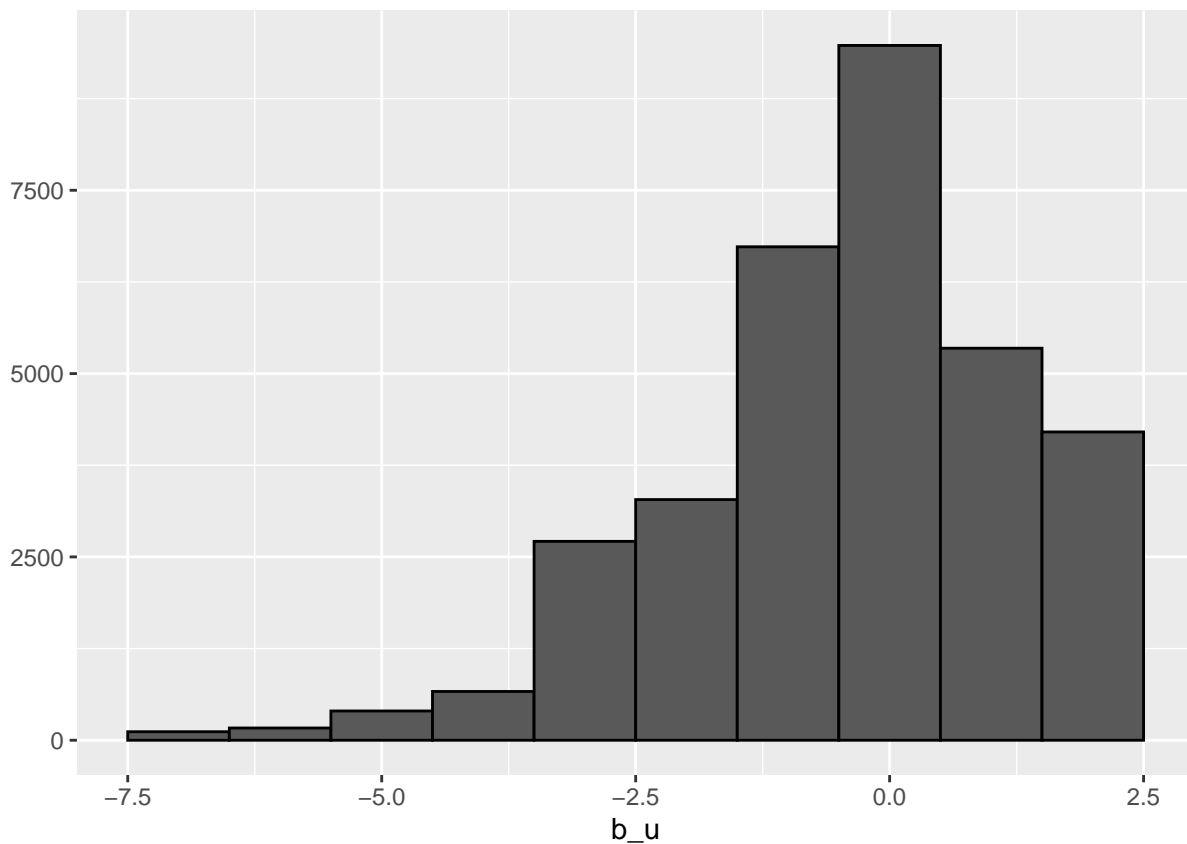
```
## [1] 7.520239
```

```
### adding user effects ###
```

```
user_avgs <- train %>%
  group_by(userid) %>%
  summarize(b_u = mean(rating - mu))
```

```
# examining the distributions of user effects
```

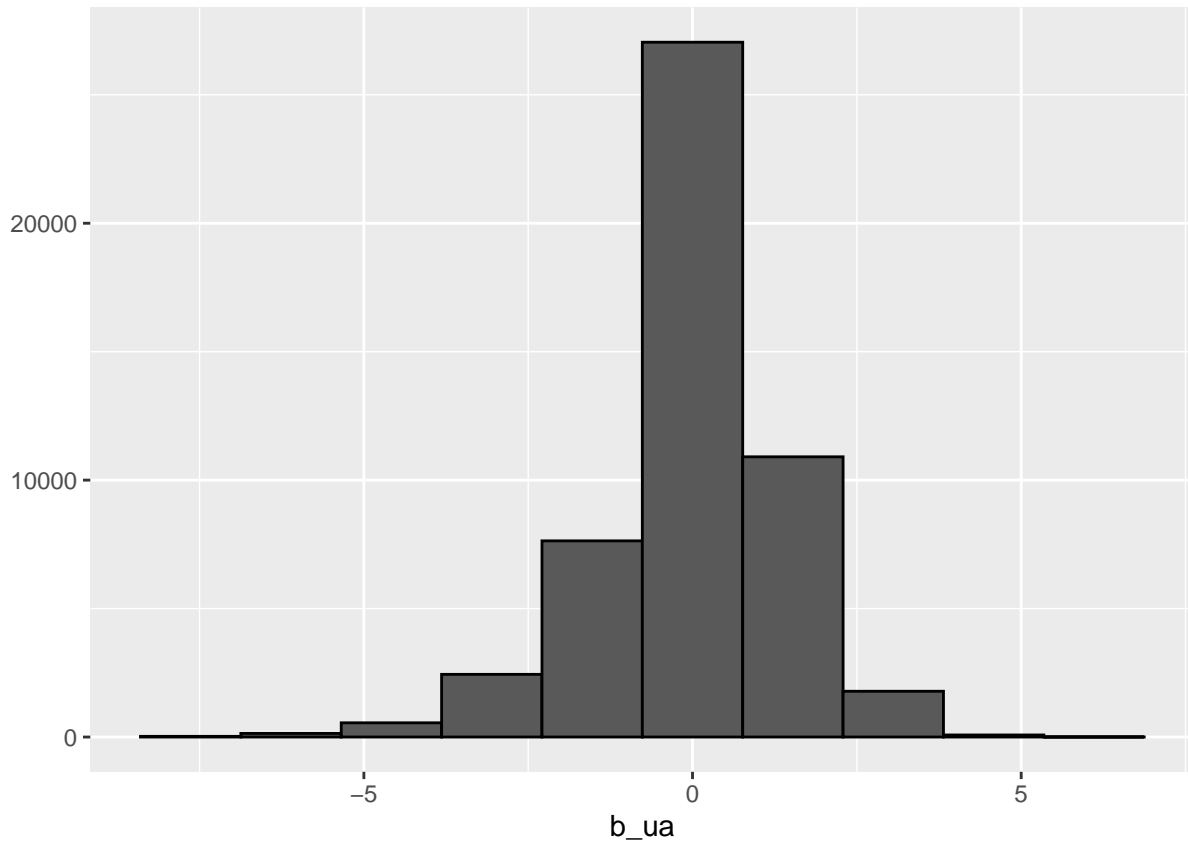
```
qplot(b_u, data = user_avgs, bins = 10, color = I("black"))
```



```
### adding userid_author effects ###
```

```
userid_author_avgs <- train %>%
  left_join(user_avgs, by='userid') %>%
  group_by(userid_author) %>%
  summarize(b_ua = mean(rating - mu - b_u))
```

```
# examining the distributions of userid_author effects
qplot(b_ua, data = userid_author_avgs, bins = 10, color = I("black"))
```



```
# checking the rmse
predicted_ratings <- test %>%
  left_join(user_avgs, by='userid') %>%
  left_join(userid_author_avgs, by='userid_author') %>%
  mutate(pred = mu + b_u + b_ua) %>%
  pull(pred)
user_and_userid_author_effects_rmse<-RMSE(predicted_ratings, test$rating, na.rm = T)

rmse_results <- rbind.data.frame(rmse_results, c("Test set: user and User*Author effects", round(user_and_userid_author_effects_rmse, 4)),
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##               method    RMSE
## 1      Just the average 1.86118
## 2           User effects 1.69647
## 3 User and age_bracket effects 1.69647
## 4      User and book effects 1.87867
## 5 User and country effects 1.69647
## 6      User and author effects 1.71147
## 7      User and year effects 1.69625
## 8      User and publisher effects 1.69914
## 9      User and User*Author effects 1.64854
```

```
## 10      User and userid_author and author effects 1.64854
## 11 User and User*Author effects with regularization 1.6516
## 12                                     Popular model 4.86484
## 13                                     Singular Value Decomposition 4.25615
## 14                                     Funk Singular Value Decomposition 4.38473
## 15      Test set: user and User*Author effects 1.63248
```

8. Conclusion

Section 8 - Conclusion and discussion

My conclusion is that under certain conditions, simple models can outperform more advanced ones. There could be several reasons for this, in this case. First, the dataset was rather small. Second, there was a small number of ratings per user and a small number of ratings per book. This could be problematic for algorithms that rely on the associations between ratings given by users (e.g. two user give similar ratings) and the associations of ratings given to items (e.g. two books receive similar ratings). Third, the matrix was extremely sparse, and that might be too difficult, even for the SVD and SVDF algorithms. Finally, I may have made a mistake in the analysis, although I tried pretty hard not to do that :). Further analyses would be required in order to find out what combination of these possibilities is indeed the cause of the poor performance of the advanced models in comparison to the simple ones.

Personally, I feel like I learned an important lesson - sometimes there is no trade-off between simplicity and accuracy. 'Simple' could sometimes be more accurate!