Choose your own project: Video Games Sales

Axel Comparetto-Berthier

18/08/2020

I)Introduction

This report marks the end of the "Choose Your Own Project" included in the HarvardX Data Science course PH125.9. Please note that english is not my native language, so there may be errors in this report. I must also mention that the code for this project was created with R 4.0.2 in case you want to run it.

I)A)Dataset description

For this project, I selected the Video Games Sales dataset available on Kaggle at the following URL: https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings. This dataset features pieces information for circa 17,000 video games, with around 7,000 games having all the 16 fields completed. These fields are:

- The game name
- The platform on which the game was released (note that for games released on several platforms there is a row for each platform)
- The game's release year
- The publisher and the developer of the game
- The genre of the game (among 12 genres)
- The Entertainment Software Rating Board (ESRB)'s rating of the game, indicating which audience the game is suited for

NB: The platform, publisher, genre, year of release and ESRB rating are referred in this report as "environmental variables".

- The mean review scores of the game for both critics and users (and the number of reviews) based on the Metacritic website
- The number of million units sold in each region (North America, European Union, Japan and the rest of the world)
- The number of million units sold worldwide

I)B)Aim of this project and key steps

The aim of this project is to develop an algorithm that will be able to predict the number of units of a game sold worldwide, given the environmental variables and the Metacritic reviews. This may seem like a useless scientific challenge, but the video game industry is actually weighting more 32,000 jobs worldwide, with revenues over 82 billions USD in 2013 (which is twice as much as the cinematographic industry) and growing ever since (source: Wikipedia).

Key steps performed included data loading and processing, the fitting of a linear model (used alongside with regularization) and a prediction using the K-nearest-neighbors algorithm (alongside with cross-validation). I ultimately wanted to see if I could have the best of both models.

I)C)Measurement metric

To measure how well the algorithm performed, I must foremost chose a metric of evaluation. This problem is a regression problem, so the most adapted metric is usually the root mean square error (RMSE). However, using the absolute error (i.e. the error in number of copies) is not a good solution here, because of the scale difference in sales for different games. This would lead to small relative errors on well-sold games weight more than big relative errors on games with a small amount of units sold.

Instead I selected to use a relative RMSE, to be computed as below, with N being the number of games and i the game index:

$$error_i = \frac{(sales_i - prediction_i)}{sales_i}$$

Note that the sign of the error is irrelevant, because we use squared error to compute RMSE.

$$RMSE = \sqrt{\frac{1}{N}\sum_{[1,N]}(error_i^2)}$$

Here is a trivial example with only two games:

```
##      gameID sales predictions error_abs error_rel
## [1,]      1  1000        1200      -200      -0.2
## [2,]      2   100          50        50       0.5
```

```
## [1] "Absolute RMSE"
```

```
## [1] 145.7738
```

```
## [1] "Relative RMSE"
```

```
## [1] 0.3807887
```

Here, the difference of scale make it seems like the prediction for game 1 is worse than the one for game 2, although it is relatively better. Furthermore, when dealing with millions of copies, absolute RMSE can be a bit inconvenient since the numbers will be way bigger than the example.

The relative RMSE seemed to be a better metric for measuring the efficiency of the algorithm.

II)Methods and analysis

II)A)Data loading, cleaning and processing

The first step of this project is the installation and activation of the required libraries:

```
###download required libraries
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

###enabling the libraries
library(tidyverse)
library(caret)
library(dplyr)
library(readr)
```

Then one can load the dataset. Since the dataset is relatively small (1.54 MB) csv file, I chose to upload it to my own Github repository and use the read_csv function of the package readr to load it. Here is the head of the dataset:

```
###download the dataset
#data originally found here : https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings
#uploaded to my github repository
url <- "https://raw.githubusercontent.com/AxelComparetto/RdataSciennce_VideoGameRating/master/Video_Game
VideoGames <- read_csv(url)
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   Platform = col_character(),
##   Year_of_Release = col_character(),
##   Genre = col_character(),
##   Publisher = col_character(),
##   NA_Sales = col_double(),
##   EU_Sales = col_double(),
##   JP_Sales = col_double(),
##   Other_Sales = col_double(),
##   Global_Sales = col_double(),
##   Critic_Score = col_double(),
##   Critic_Count = col_double(),
##   User_Score = col_character(),
##   User_Count = col_double(),
##   Developer = col_character(),
##   Rating = col_character()
## )
```

```
head(VideoGames)
```

```
## # A tibble: 6 x 16
##   Name  Platform Year_of_Release Genre Publisher NA_Sales EU_Sales JP_Sales
##   <chr> <chr>    <chr>           <chr> <chr>        <dbl>    <dbl>    <dbl>
## 1 Wii ~ Wii      2006            Spor~ Nintendo      41.4    29.0      3.77
## 2 Supe~ NES      1985            Plat~ Nintendo      29.1     3.58     6.81
## 3 Mari~ Wii      2008            Raci~ Nintendo      15.7    12.8      3.79
## 4 Wii ~ Wii      2009            Spor~ Nintendo      15.6    10.9      3.28
## 5 Poke~ GB       1996            Role~ Nintendo      11.3     8.89    10.2
## 6 Tetr~ GB       1989            Puzz~ Nintendo      23.2     2.26     4.22
## # ... with 8 more variables: Other_Sales <dbl>, Global_Sales <dbl>,
## #   Critic_Score <dbl>, Critic_Count <dbl>, User_Score <chr>, User_Count <dbl>,
## #   Developer <chr>, Rating <chr>
```

For this study, I do not take the games with missing information into account, thus the use of na.omit. I then created a "gamedId" column and changed some scales for more clarity:

```
###data cleaning
VideoGames <- na.omit(VideoGames) #clear the rows with missing values
VideoGames <- VideoGames %>% mutate(gameId = which(Name == Name)) #create a gameId column
vec_reorder <- c(17, 1:16) #a vector to place gameId as first column
VideoGames <- VideoGames %>% select(all_of(vec_reorder)) #places gameId as first column

###changing scales
#putting User_score (/10) and Critic_score(/100) on the same scale (/100)
```

```
VideoGames <- VideoGames %>% mutate(User_Score = 10*as.numeric(User_Score))
#the original dataset is in million copies, I will use the number of units instead
VideoGames <- VideoGames %>% mutate(NA_Sales=NA_Sales*10^6,
                                    EU_Sales=EU_Sales*10^6,
                                    JP_Sales=JP_Sales*10^6,
                                    Other_Sales=Other_Sales*10^6,
                                    Global_Sales=Global_Sales*10^6)
```

The next step is to generate a training and a test set. To chose the ratio between training and test set, I looked at how many unique predictors there are.

```
#evaluate the number of unique predictors
length(unique(VideoGames$Platform))
```

```
## [1] 17
```

```
length(unique(VideoGames$Genre))
```

```
## [1] 12
```

```
length(unique(VideoGames$Year_of_Release))
```

```
## [1] 26
```

```
length(unique(VideoGames$Publisher))
```

```
## [1] 263
```

```
length(unique(VideoGames$Developer)) #too many developers to ensure every dev in test set is in trainin
```

```
## [1] 1297
```

```
length(unique(VideoGames$Rating))
```

```
## [1] 7
```

There are many publishers, so that I should not use a 50/50 data partition, to ensure there are several games of the same developer within the training set with different genres, rating and so on. This is why a partition of 80% data in the training set and 20% in the test set seemed more appropriate to me.

```
#create the data partition
set.seed(1, sample.kind = "Rounding")
indexes <- createDataPartition(y=VideoGames$gameId, times=1, p=0.2, list=FALSE)
#80% training set, 20% test set
training <- VideoGames[-indexes,]
temp <- VideoGames[indexes,]
#make sure every platform/genre/YoR/Publisher/Rating in the test set is in the training set
test <- temp %>%
```

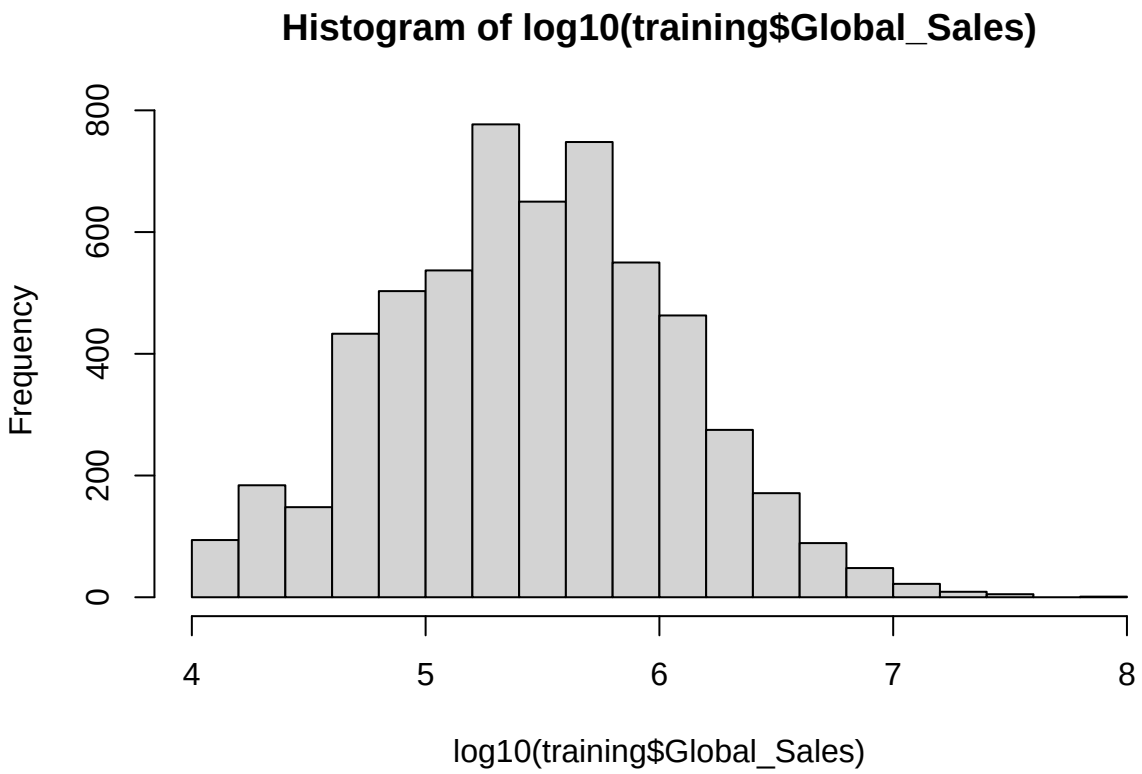
```
  semi_join(training, by = "Platform") %>%
  semi_join(training, by = "Genre") %>%
  semi_join(training, by = "Year_of_Release") %>%
  semi_join(training, by = "Publisher") %>%
  semi_join(training, by = "Rating") %>%
  semi_join(training, by = "Developer")
removed <- anti_join(temp, test)
```

```
## Joining, by = c("gameId", "Name", "Platform", "Year_of_Release", "Genre", "Publisher", "NA_Sales", "
```

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

###free memory
rm(url, vec_reorder, temp, removed, indexes)
```

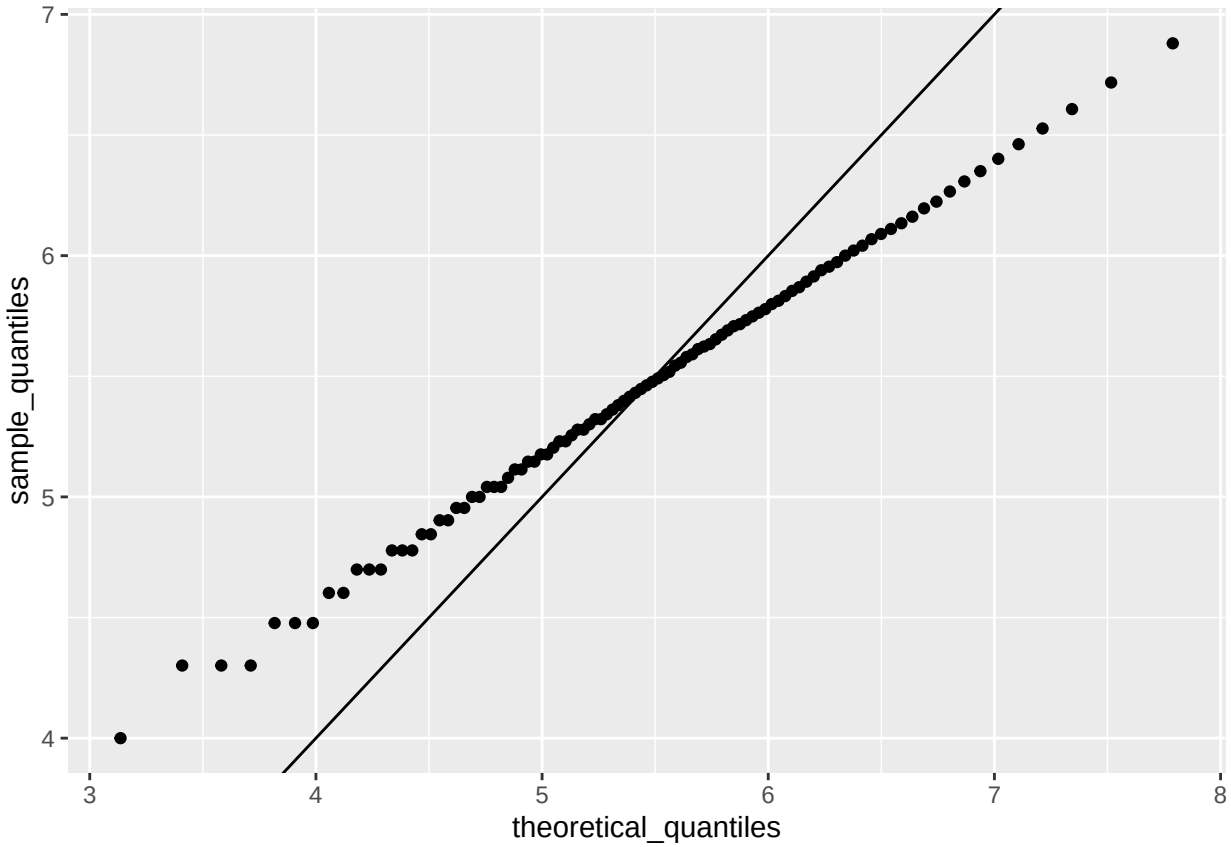
II)B)Naive method

I wanted to have an insight of the distribution of the global sales in the training values, so I used histograms.



With these two histograms, I realized that working with the raw amount of copies sold was not convenient, so I tried to work with the log. Furthermore, my basic knowledge about video games sales is that the effects of biases on the sales are likely to be multiplicative, rather than additive: the first histogram shows the difference of scale between the least and the most sold games is better expressed with a ration, not with a difference. This is what we observe in reality, because AAA games are sold many more times than small games by independent studios.

The second histogram shows that the distribution of the log_{10} of the number of copies sold is somewhat close to normal approximation, which calls for a linear model. Here is a qqplot:



Now that I selected the data partition, the measurement metric and the scale I want to work with, I wanted to have some reference I could compare my models to, so I ran the simplest of predictions: predicting that every game has sold the average amount of copies. This average is referred to as μ.

```
#defining how to compute the RMSE
fun_RMSE <- function(actual_values, predictions)
{
  percent_error = (actual_values-predictions)/actual_values
  sqrt(mean(percent_error^2))
}

log10_mu <- mean(training$logGlobal_Sales) #overall mean of the log10 sales worldwide

#making naive predictions: every game sold the average number of copies
pred = 10^log10_mu
RMSE <- fun_RMSE(test$Global_Sales, pred)
#creating a matrix to keep track of successive RMSEs
mat_RMSE <- matrix(data = c("Worldwide naive approach: average", RMSE), nrow = 1, ncol = 2)
colnames(mat_RMSE) <- c("Method", "RMSE")
```

I stored the RMSE for each method in a matrix. Here is the first row:

```
##      Method                                RMSE
## [1,] "Worldwide naive approach: average" "3.83191547183535"
```

The naive method has an RMSE of about 3.83. Since the error used to compute the RMSE is a percentage,

the RMSE is also a percentage, meaning that the naive method's prediction is 383% off, which is a lot of course. Let's see if I can improve it.

II)C)Using a linear model

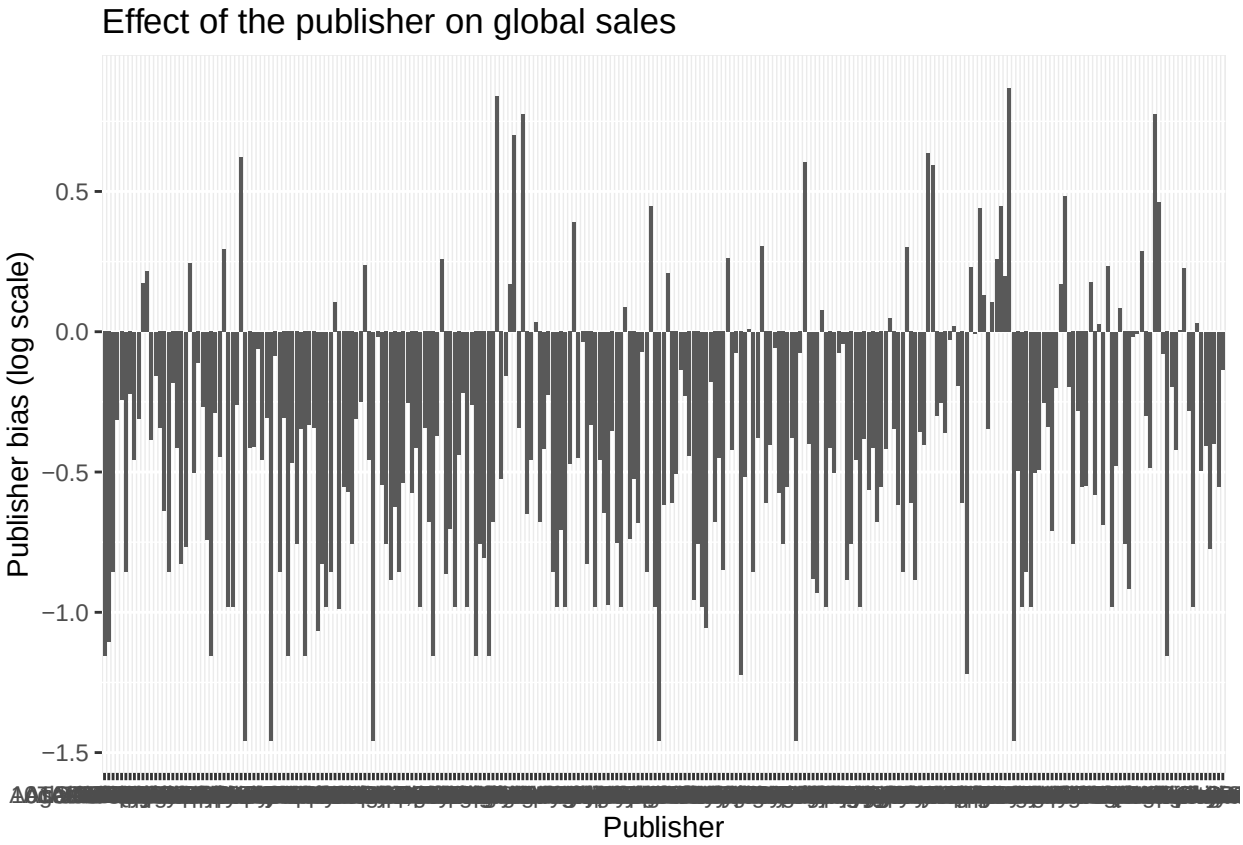
The first idea I had was a linear model: I would use the biases for each environmental variable and the reviews to predict global sales.

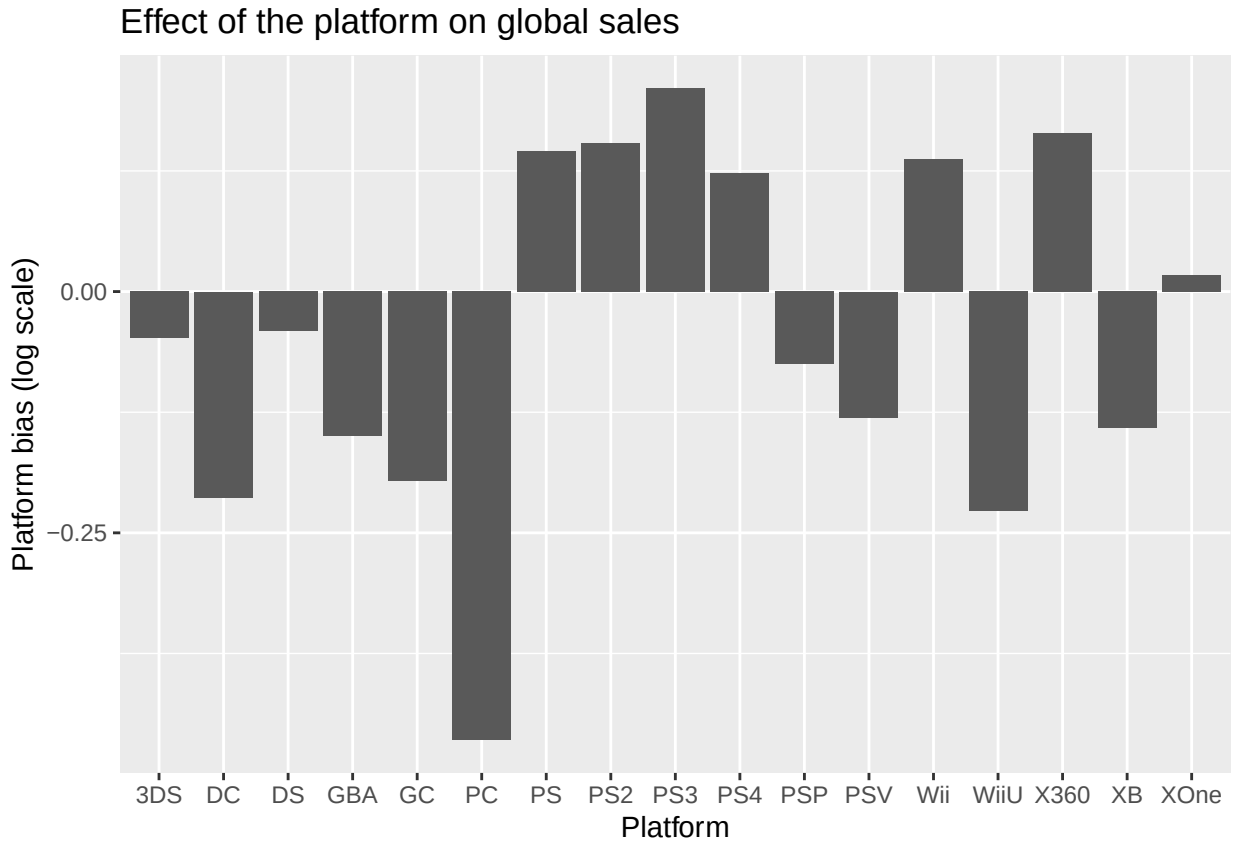
II)C)1)Using environmental bias

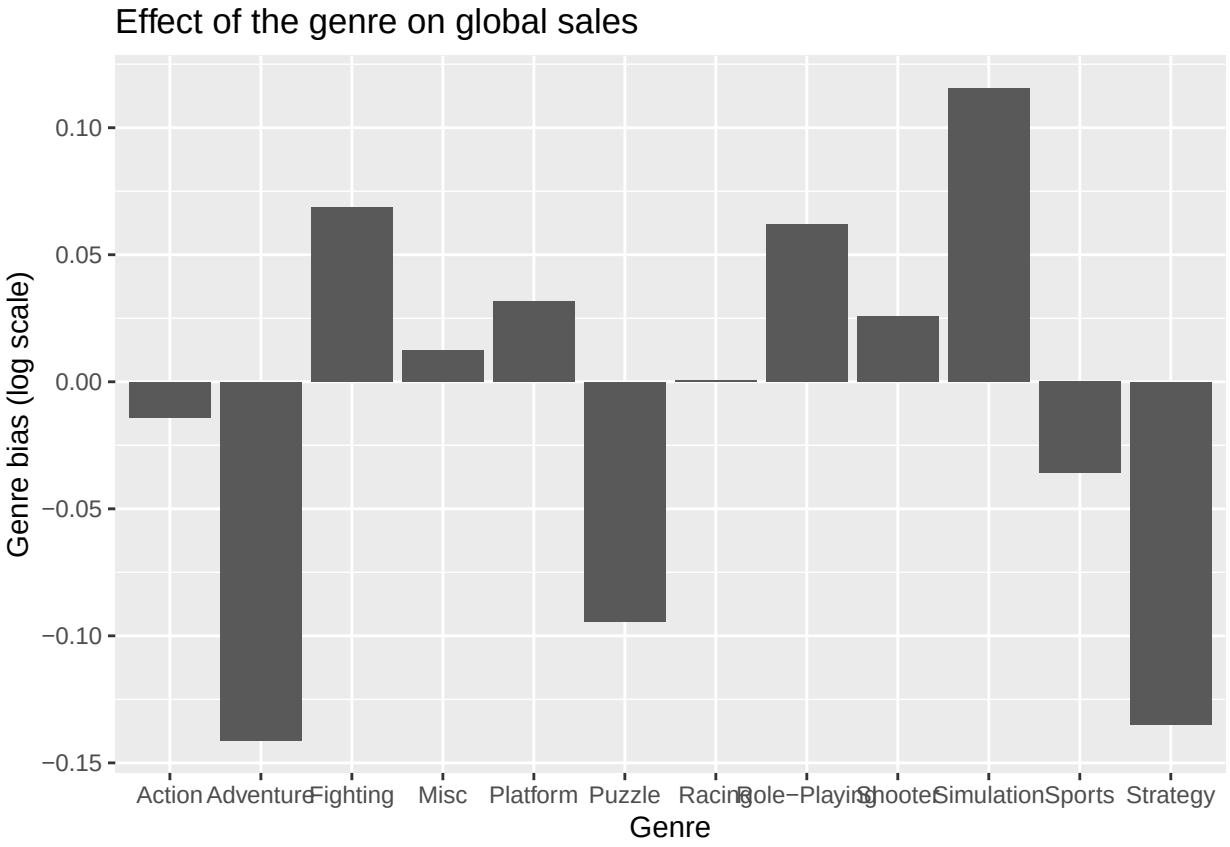
I started with the environmental biases. I evaluated each bias in the following order: Publisher, Platform, Genre, Year of release, Rating and Developer. The order is irrelevant on the final result but changes slightly the evaluation of each bias, since the evaluation of a bias takes the biases already evaluated into account. The biases are already computed in log scale Here is the code for the evaluation of the biases:

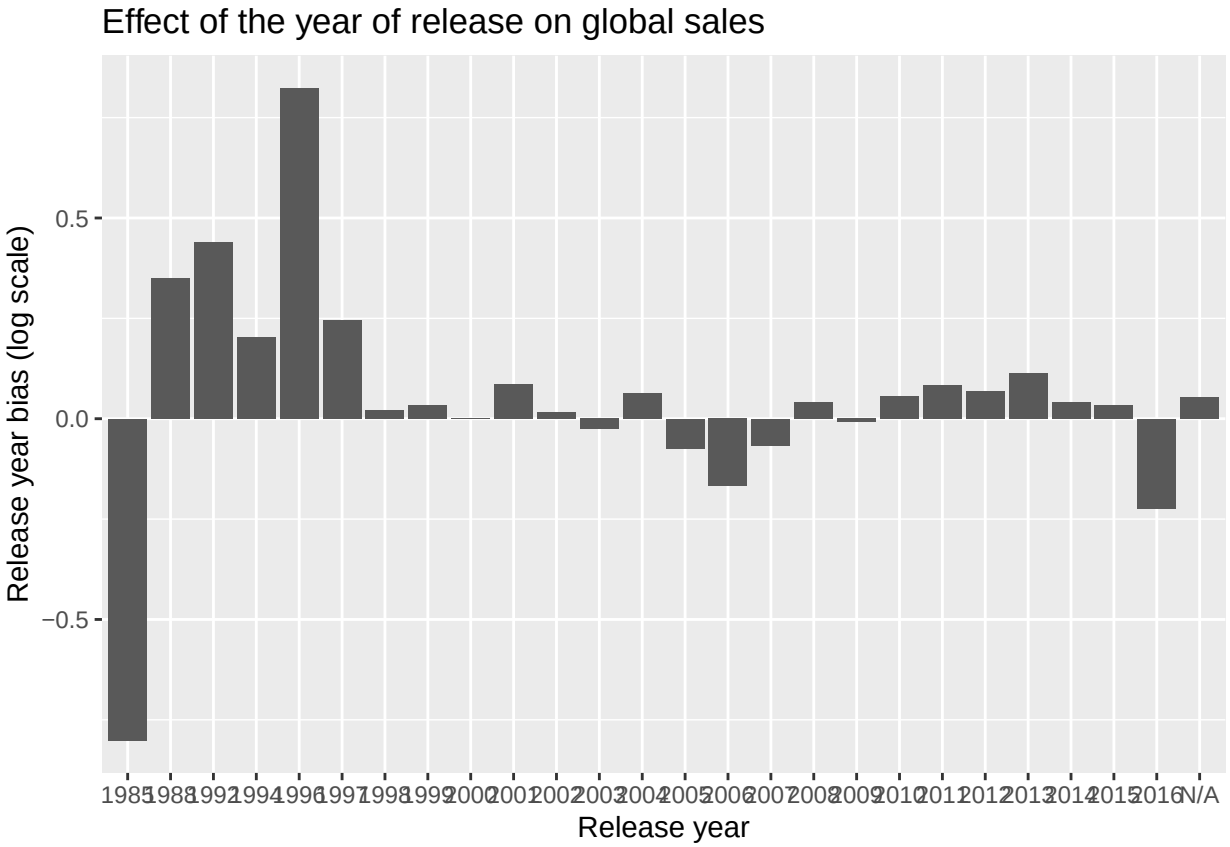
```
###publisher bias
bias_pub <- training %>%
  group_by(Publisher) %>%
  summarise(b_pu  = mean(logGlobal_Sales - log10_mu),
            count_pu = n(),
            sum_bpulog = sum(logGlobal_Sales - log10_mu),
            .groups="drop")

###platform bias
bias_plat <- training %>%
  left_join(bias_pub, by="Publisher") %>%
  group_by(Platform) %>%
  summarise(b_pla  = mean(logGlobal_Sales -log10_mu - b_pu),
            count_pla = n(),
            sum_bplalog = sum(logGlobal_Sales - log10_mu - b_pu),
            .groups="drop")

###genre bias
bias_genre <- training %>%
  left_join(bias_pub, by="Publisher") %>%
  left_join(bias_plat, by="Platform") %>%
  group_by(Genre) %>%
  summarise(b_ge  = mean(logGlobal_Sales - log10_mu - b_pu - b_pla),
            count_ge = n(),
            sum_bgelog = sum(logGlobal_Sales - log10_mu - b_pu - b_pla),
            .groups="drop")

###year of release bias
bias_yor <- training %>%
  left_join(bias_pub, by="Publisher") %>%
  left_join(bias_plat, by="Platform") %>%
  left_join(bias_genre, by="Genre") %>%
  group_by(Year_of_Release) %>%
  summarise(b_yor  = mean(logGlobal_Sales - log10_mu - b_pu - b_pla - b_ge),
            count_yor = n(),
            sum_byorlog = sum(logGlobal_Sales - log10_mu - b_pu - b_pla - b_ge),
            .groups="drop")
```

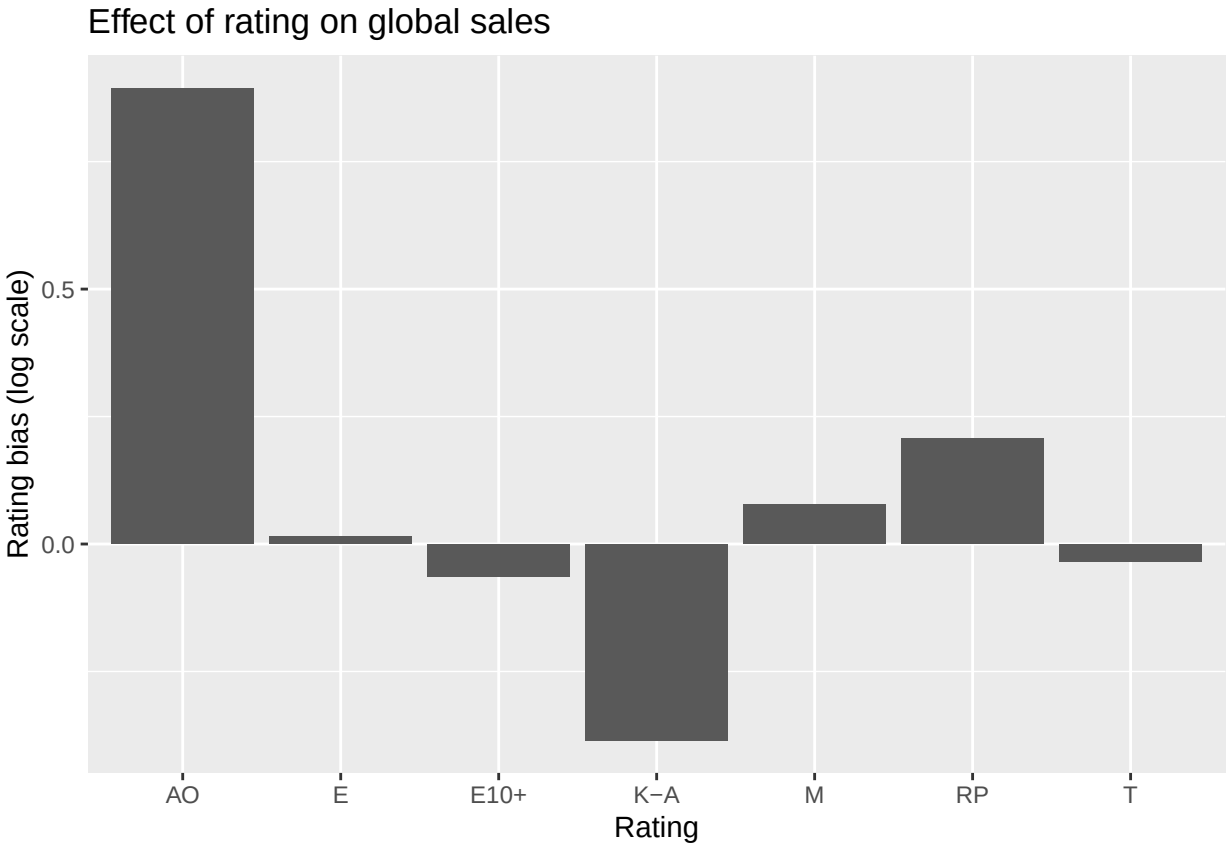
```r
###rating bias
bias_rat <- training %>%
  left_join(bias_pub, by="Publisher") %>%
  left_join(bias_plat, by="Platform") %>%
  left_join(bias_genre, by="Genre") %>%
  left_join(bias_yor, by="Year_of_Release") %>%
  group_by(Rating) %>%
  summarise(b_rat  = mean(logGlobal_Sales - log10_mu - b_pu - b_pla - b_ge - b_yor),
            count_rat = n(),
            sum_bratlog = sum(logGlobal_Sales - log10_mu - b_pu - b_pla - b_ge - b_yor),
            .groups="drop")

###developer bias
bias_dev <- training %>%
  left_join(bias_pub, by="Publisher") %>%
  left_join(bias_plat, by="Platform") %>%
  left_join(bias_genre, by="Genre") %>%
  left_join(bias_yor, by="Year_of_Release") %>%
  left_join(bias_rat, by="Rating") %>%
  group_by(Developer) %>%
  summarise(b_dev  = mean(logGlobal_Sales - log10_mu - b_pu - b_pla - b_ge - b_yor - b_rat),
            count_dev = n(),
            sum_bdevlog = sum(logGlobal_Sales - log10_mu - b_pu - b_pla - b_ge - b_yor - b_rat),
            .groups="drop")

###computes the bias fro each row of the test set
vec_bias_pub <- sapply(test$Publisher, function(x)
  bias_pub$b_pu[which(bias_pub$Publisher == x)])
vec_bias_pla <- sapply(test$Platform, function(x)
  bias_plat$b_pla[which(bias_plat$Platform == x)])
vec_bias_ge <- sapply(test$Genre, function(x)
  bias_genre$b_ge[which(bias_genre$Genre == x)])
vec_bias_yor <- sapply(test$Year_of_Release, function(x)
  bias_yor$b_yor[which(bias_yor$Year_of_Release == x)])
vec_bias_rat <- sapply(test$Rating, function(x)
  bias_rat$b_rat[which(bias_rat$Rating == x)])
vec_bias_dev <- sapply(test$Developer, function(x)
  bias_dev$b_dev[which(bias_dev$Developer == x)])
```

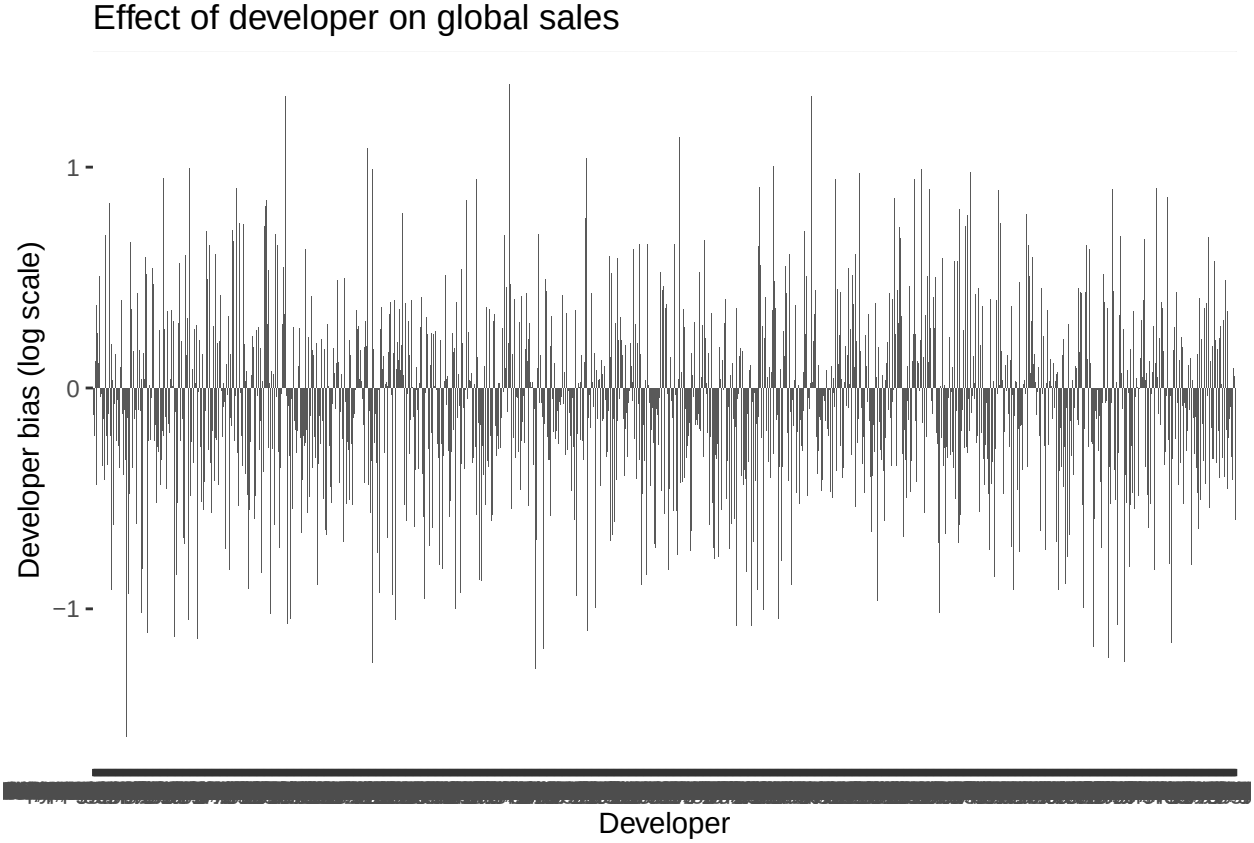

Effect of the publisher on global sales
Publisher bias (log scale)
0.5
0.0
−0.5
−1.0
−1.5
Publisher


Effect of the platform on global sales
Platform bias (log scale)
0.00
−0.25
3DS
DC
DS
GBA
GC
PC
PS
PS2
PS3
PS4
PSP
PSV
Wii
WiiU
X360
XB
XOne
Platform


Effect of the genre on global sales
Genre bias (log scale)
0.10
0.05
0.00
−0.05
−0.10
−0.15
Action
Adventure
Fighting
Misc
Platform
Puzzle
Racing
Role−Playing
Shooter
Simulation
Sports
Strategy
Genre


Effect of the year of release on global sales
Release year bias (log scale)
0.5
0.0
−0.5
1985
1988
1992
1994
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
N/A
Release year


Effect of rating on global sales
Rating bias (log scale)
0.5
0.0
AO
E
E10+
K−A
M
RP
T
Rating



The graphs, associated with the computation of the mean and standard deviation of the biases, are helping to know whether a predictor is relevant or not: if the absolute value of the mean is less than the standard deviation, it means that the predictor has high variability, that must be considered.

```
#publisher bias mean and sd
mean(bias_pub$b_pu)
```

```
## [1] -0.4318683
```

```
sd(bias_pub$b_pu)
```

```
## [1] 0.4707106
```

```
#platform bias mean and sd
mean(bias_plat$b_pla)
```

```
## [1] -0.04315546
```

```
sd(bias_plat$b_pla)
```

```
## [1] 0.1836938
```

```r
#genre bias mean and sd
mean(bias_genre$b_ge)
```

```
## [1] -0.008646128
```

```r
sd(bias_genre$b_ge)
```

```
## [1] 0.08052758
```

```r
#year of release bias mean and sd
mean(bias_yor$b_yor)
```

```
## [1] 0.0541151
```

```r
sd(bias_yor$b_yor)
```

```
## [1] 0.2679516
```

```r
#rating bias mean and sd
mean(bias_rat$b_rat)
```

```
## [1] 0.1013021
```

```r
sd(bias_rat$b_rat)
```

```
## [1] 0.393568
```

```r
#developer bias mean and sd
mean(bias_dev$b_dev)
```

```
## [1] -0.07243421
```

```r
sd(bias_dev$b_dev)
```

```
## [1] 0.4053816
```

Using this criteria, it seems like all the predictors are relevant, with publisher being the most relevant and genre the least relevant. It calls for a linear model like this:

$$log_{10}(prediction) = log_{10}(\mu) + b_{pub} + b_{pla} + b_{ge} + b_{yor} + b_{rat} + b_{dev}$$

With μ being the average amount of copies sold, b the bias, respectively for Publisher, Platform, Genre, Year of release, Rating and Developer, and α β and γ being constants that optimize the linear model predictions. This linear model states that the biases have a multiplicative effect on global sales, due to the log property:

$$prediction = 10^{log_{10}(\mu)+b_{pub}+b_{pla}+b_{ge}+b_{yor}+b_{rat}+b_{dev}}$$

$$prediction = \mu \times 10^{b_{pub}} \times 10^{b_{pla}} \times 10^{b_{ge}} \times 10^{b_{yor}} \times 10^{b_{rat}} \times 10^{b_{dev}}$$

Let's use this model to make predictions:

```r
#using the four bias to make predictions
pred = 10^(log10_mu + vec_bias_pub + vec_bias_pla + vec_bias_ge +
             vec_bias_yor + vec_bias_rat + vec_bias_dev)
RMSE <- fun_RMSE(test$Global_Sales, pred)
#keep track of RMSE
mat_RMSE <- rbind(mat_RMSE, c("LM: Worldwide with environmental bias (publisher, platform, genre, YoR, r
```

```
##      Method
## [1,] "Worldwide naive approach: average"
## [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
##      RMSE
## [1,] "3.83191547183535"
## [2,] "2.86489546914096"
```

The RMSE is now 2.86, which is better but still far from real sales.

The use of regularization may help a little (for instance, a lot of developers in this set only produced one game and for some of them the bias is quite high). The code below regularize each bias successively, taking the bias already regularized into account:

```r
###computes the bias for the rows of the training set before regularizing
t_v_b_pub <- sapply(training$Publisher, function(x)
  bias_pub$b_pu[which(bias_pub$Publisher == x)])
t_v_b_pla <- sapply(training$Platform, function(x)
  bias_plat$b_pla[which(bias_plat$Platform == x)])
t_v_b_ge <- sapply(training$Genre, function(x)
  bias_genre$b_ge[which(bias_genre$Genre == x)])
t_v_b_yor <- sapply(training$Year_of_Release, function(x)
  bias_yor$b_yor[which(bias_yor$Year_of_Release == x)])
t_v_b_rat <- sapply(training$Rating, function(x)
  bias_rat$b_rat[which(bias_rat$Rating == x)])
t_v_b_dev <- sapply(training$Developer, function(x)
  bias_dev$b_dev[which(bias_dev$Developer == x)])

###regularization of publisher bias
lambdas <- seq(0, 100, 1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_pub_bias <- bias_pub$sum_bpulog/(lam + bias_pub$count_pu) #regularized publisher bias
  t_v_b_pub_reg <- sapply(training$Publisher, function(x)
    reg_pub_bias[which(bias_pub$Publisher == x)])
  pred <- 10^(log10_mu + t_v_b_pub_reg +
                t_v_b_pla + t_v_b_ge +
                t_v_b_yor + t_v_b_rat + t_v_b_dev)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
lambda_pub <- lambdas[which.min(RMSEs_train)]
#creating a new column for regularized publisher bias
bias_pub <- bias_pub %>% mutate(reg_b_pub  = sum_bpulog/(lambda_pub + count_pu))
#re compute the publisher bias for training set row with regularized data
t_v_b_pub <- sapply(training$Publisher, function(x)
  bias_pub$reg_b_pub[which(bias_pub$Publisher == x)])
```

```
###regularization of platform bias
lambdas <- seq(0, 100, 1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_pla_bias <- bias_plat$sum_bplalog/(lam + bias_plat$count_pla) #regularized platform bias
  t_v_b_pla_reg <- sapply(training$Platform, function(x)
    reg_pla_bias[which(bias_plat$Platform == x)])
  pred <- 10^(log10_mu + t_v_b_pub +
                t_v_b_pla_reg + t_v_b_ge +
                t_v_b_yor + t_v_b_rat + t_v_b_dev)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
lambda_plat <- lambdas[which.min(RMSEs_train)]
#creating a new column for regularized platform bias
bias_plat <- bias_plat %>% mutate(reg_b_pla  = sum_bplalog/(lambda_plat + count_pla))
#re compute the platform bias for training set row with regularized data
t_v_b_pla <- sapply(training$Platform, function(x)
  bias_plat$reg_b_pla[which(bias_plat$Platform == x)])

###regularization of genre bias
lambdas <- seq(0, 100, 1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_ge_bias <- bias_genre$sum_bgelog/(lam + bias_genre$count_ge) #regularized genre bias
  t_v_b_ge_reg <- sapply(training$Genre, function(x)
    reg_ge_bias[which(bias_genre$Genre == x)])
  pred <- 10^(log10_mu + t_v_b_pub +
                t_v_b_pla + t_v_b_ge_reg +
                t_v_b_yor + t_v_b_rat + t_v_b_dev)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
lambda_ge <- lambdas[which.min(RMSEs_train)]
#creating a new column for regularized genre bias
bias_genre <- bias_genre %>% mutate(reg_b_ge  = sum_bgelog/(lambda_ge + count_ge))
#re compute the genre bias for training set row with regularized data
t_v_b_ge <- sapply(training$Genre, function(x)
  bias_genre$reg_b_ge[which(bias_genre$Genre == x)])

###regularization of year of release bias
lambdas <- seq(20, 30, 1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_yor_bias <- bias_yor$sum_byorlog/(lam + bias_yor$count_yor) #regularized year of release bias
  t_v_b_yor_reg <- sapply(training$Year_of_Release, function(x)
    reg_yor_bias[which(bias_yor$Year_of_Release == x)])
  pred <- 10^(log10_mu + t_v_b_pub +
                t_v_b_pla + t_v_b_ge +
                t_v_b_yor_reg + t_v_b_rat + t_v_b_dev)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
```

```
lambda_yor <- lambdas[which.min(RMSEs_train)]
#creating a new column for regularized release year bias
bias_yor <- bias_yor %>% mutate(reg_b_yor  = sum_byorlog/(lambda_yor + count_yor))
#re compute the release year bias for training set row with regularized data
t_v_b_yor <- sapply(training$Year_of_Release, function(x)
  bias_yor$reg_b_yor[which(bias_yor$Year_of_Release  == x)])

###regularization of rating bias
lambdas <- seq(5, 100, 1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_rat_bias <- bias_rat$sum_bratlog/(lam + bias_rat$count_rat) #regularized rating bias
  t_v_b_rat_reg <- sapply(training$Rating, function(x)
    reg_rat_bias[which(bias_rat$Rating == x)])
  pred <- 10^(log10_mu + t_v_b_pub +
                t_v_b_pla + t_v_b_ge +
                t_v_b_yor + t_v_b_rat_reg + t_v_b_dev)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
lambda_rat <- lambdas[which.min(RMSEs_train)]
#creating a new column for regularized rating bias
bias_rat <- bias_rat %>% mutate(reg_b_rat  = sum_bratlog/(lambda_rat + count_rat))
#re compute the rating bias for training set row with regularized data
t_v_b_rat <- sapply(training$Rating, function(x)
  bias_rat$reg_b_rat[which(bias_rat$Rating  == x)])

###regularization of developer bias
lambdas <- seq(5, 100, 1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_dev_bias <- bias_dev$sum_bdevlog/(lam + bias_dev$count_dev) #regularized developer bias
  t_v_b_dev_reg <- sapply(training$Developer, function(x)
    reg_dev_bias[which(bias_dev$Developer == x)])
  pred <- 10^(log10_mu + t_v_b_pub +
                t_v_b_pla + t_v_b_ge +
                t_v_b_yor + t_v_b_rat + t_v_b_dev_reg)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
lambda_dev <- lambdas[which.min(RMSEs_train)]
#creating a new column for regularized rating bias
bias_dev <- bias_dev %>% mutate(reg_b_dev  = sum_bdevlog/(lambda_dev + count_dev))
#re compute the rating bias for training set row with regularized data
t_v_b_dev <- sapply(training$Developer, function(x)
  bias_dev$reg_b_dev[which(bias_dev$Developer  == x)])

###computes the regularized bias fro each row of the test set
vec_bias_pub <- sapply(test$Publisher, function(x)
  bias_pub$reg_b_pub[which(bias_pub$Publisher == x)])
vec_bias_pla <- sapply(test$Platform, function(x)
  bias_plat$reg_b_pla[which(bias_plat$Platform == x)])
vec_bias_ge <- sapply(test$Genre, function(x)
```

```
  bias_genre$reg_b_ge[which(bias_genre$Genre == x)])
vec_bias_yor <- sapply(test$Year_of_Release, function(x)
  bias_yor$reg_b_yor[which(bias_yor$Year_of_Release == x)])
vec_bias_rat <- sapply(test$Rating, function(x)
  bias_rat$reg_b_rat[which(bias_rat$Rating == x)])
vec_bias_dev <- sapply(test$Developer, function(x)
  bias_dev$reg_b_dev[which(bias_dev$Developer == x)])

###free memory
rm(lambdas, lambda_ge, lambda_plat, lambda_pub, lambda_rat,
   lambda_yor, lambda_dev, RMSEs_train, pred)
```

Let's now make the predictions and evaluate the RMSE:

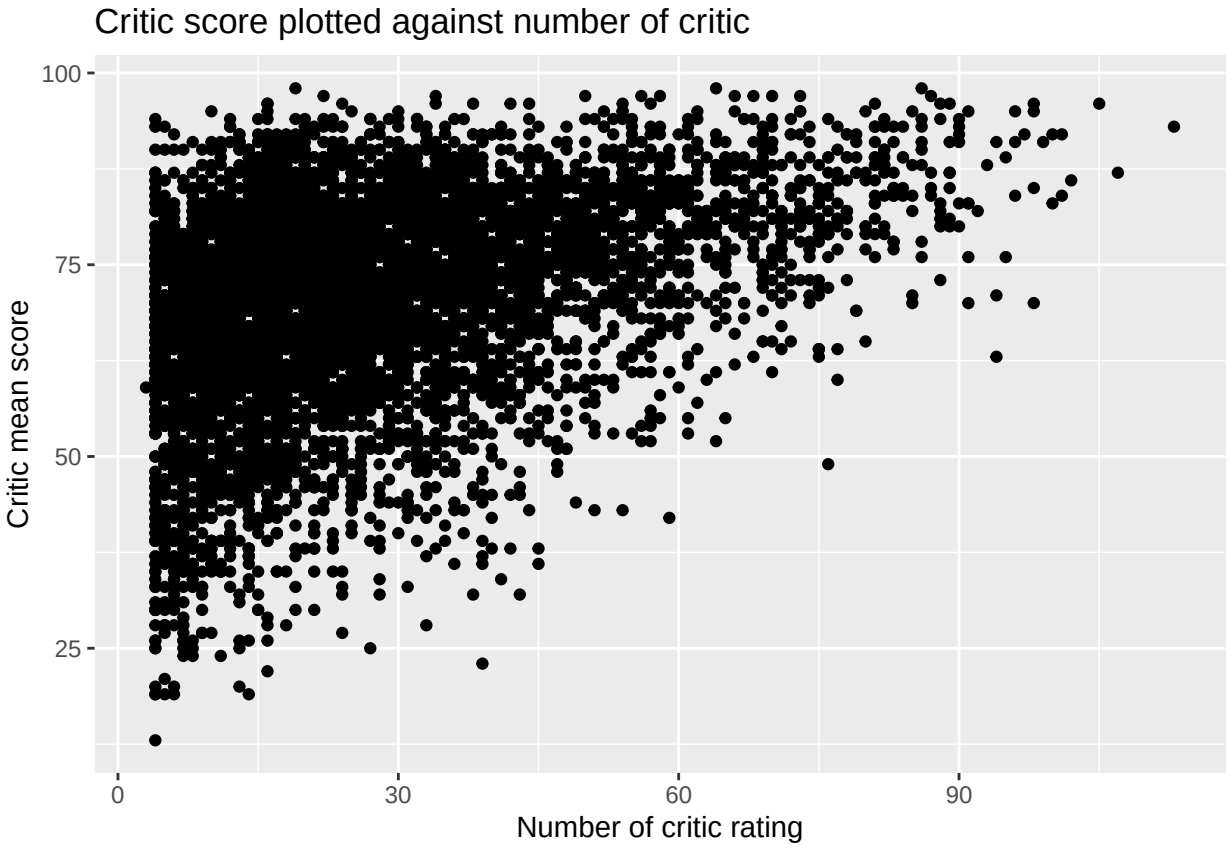
```
#using the four bias to make predictions
pred = 10^(log10_mu + vec_bias_pub + vec_bias_pla + vec_bias_ge +
             vec_bias_yor + vec_bias_rat + vec_bias_dev)
RMSE <- fun_RMSE(test$Global_Sales, pred)
#keep track of RMSE
mat_RMSE <- rbind(mat_RMSE, c("LM: Worldwide with regularized environmental bias", RMSE))
```

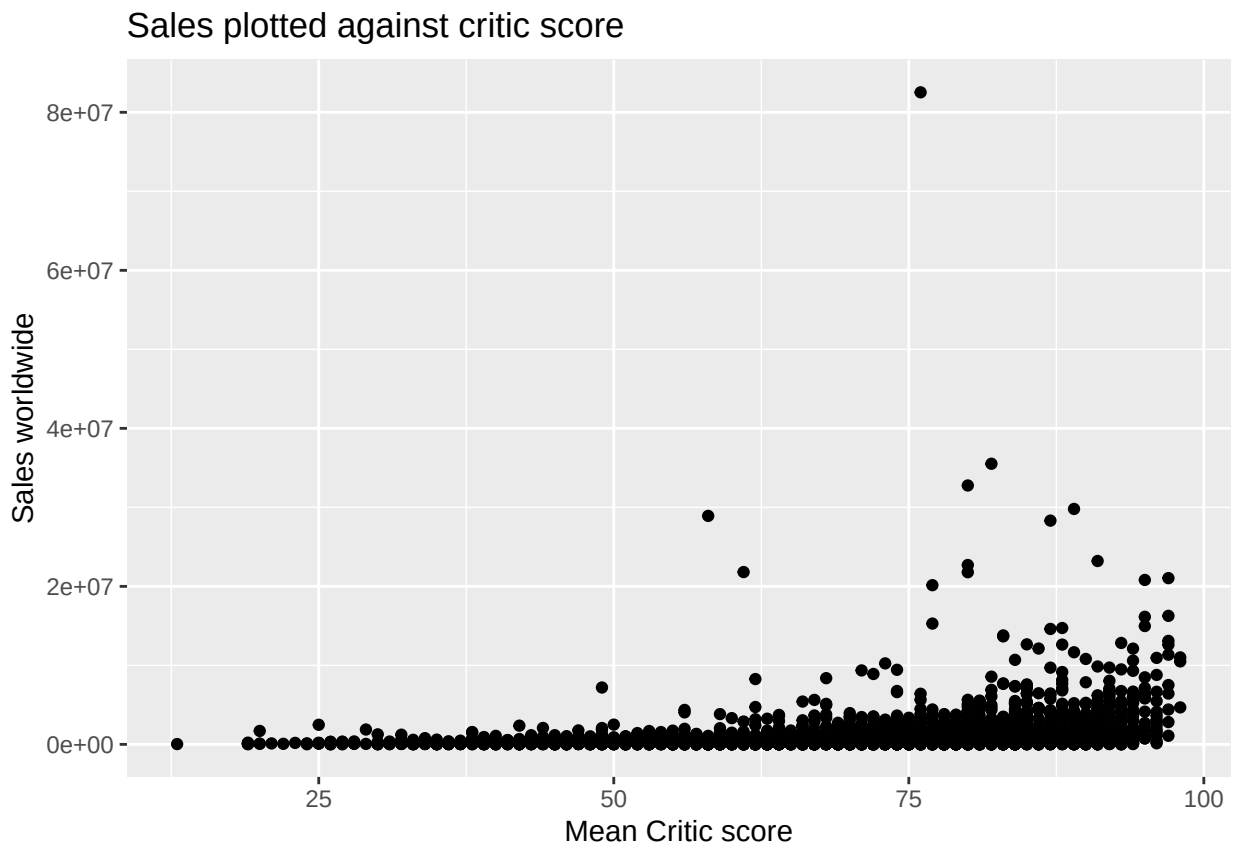
```
##      Method
## [1,] "Worldwide naive approach: average"
## [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
## [3,] "LM: Worldwide with regularized environmental bias"
##      RMSE
## [1,] "3.83191547183535"
## [2,] "2.86489546914096"
## [3,] "2.60215666537728"
```

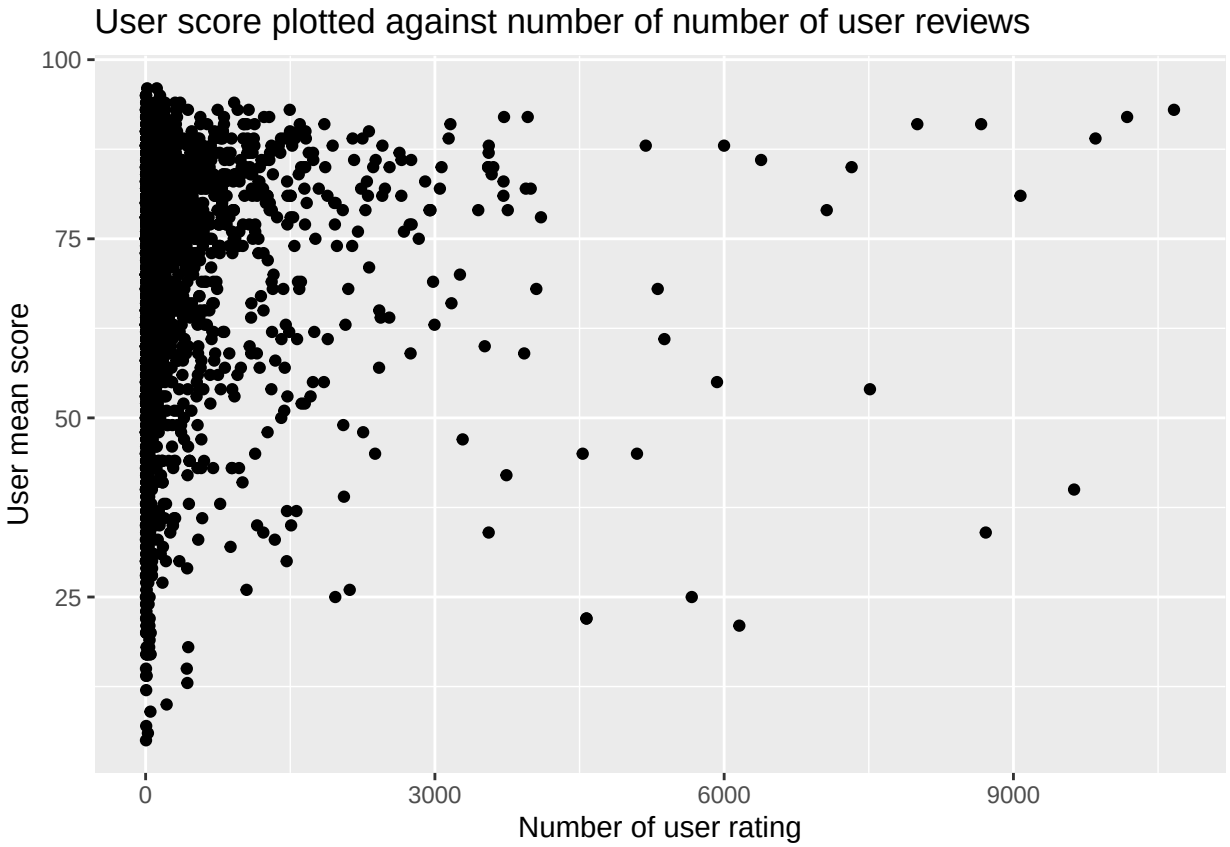
The RMSE is now around 2.60, which is slightly better. But I have not used reviews yet.

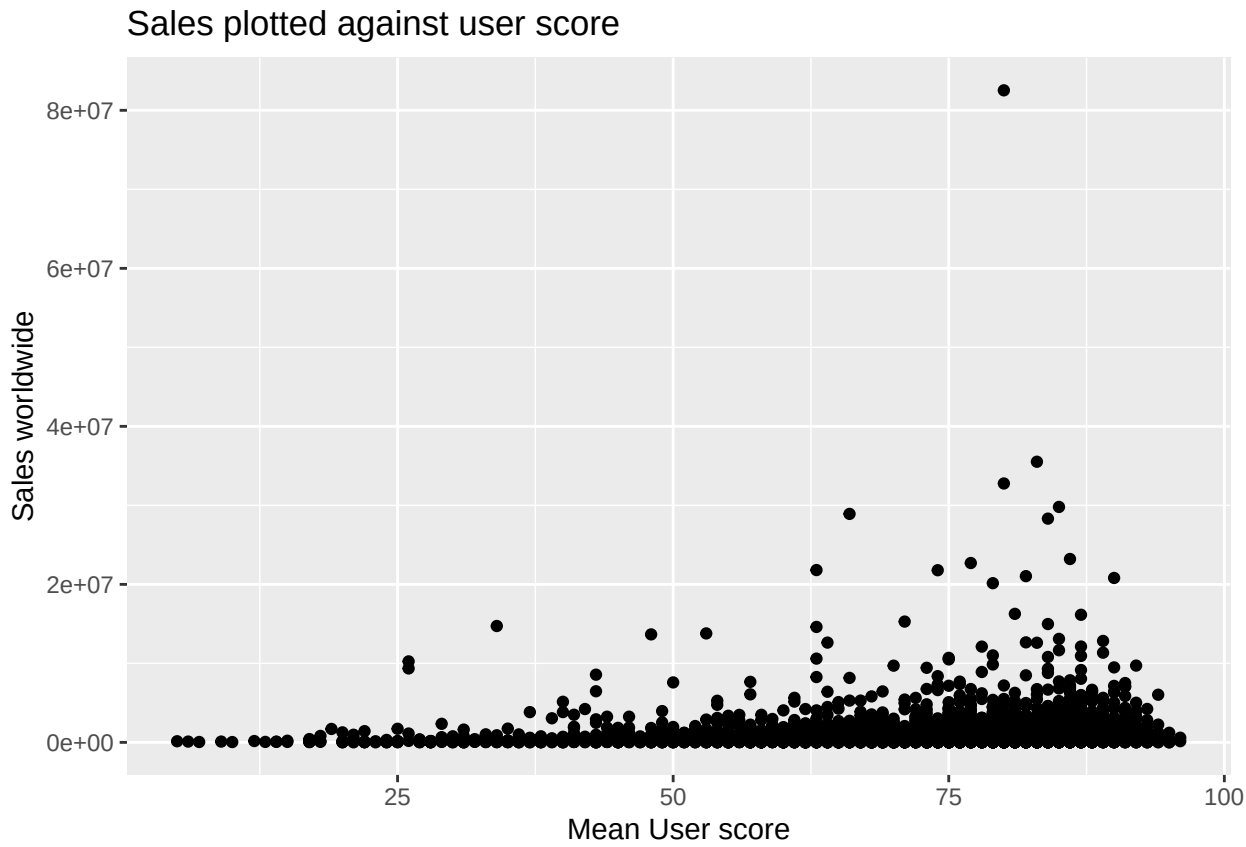
II)C)2)Using reviews

Let's first look at the data to see if a correlation exists between review score and review count, or between review score and global sales.


Critic score plotted against number of critic
Critic mean score
100
75
50
25
0
30
60
90
Number of critic rating


Sales plotted against critic score
Sales worldwide
8e+07
6e+07
4e+07
2e+07
0e+00
25
50
75
100
Mean Critic score


User score plotted against number of number of user reviews
User mean score
100
75
50
25
0
3000
6000
9000
Number of user rating



From these graphs we can see that there is no actual correlation between high scores and high sales, nor between review score and review count. Also, there are a lot of games with high score and low review count, which calls for regularization later. Let's first evaluate the impact of review through a linear model:

$$log_{10}(prediction) = log_{10}(\mu) + b_{pub} + b_{pla} + b_{ge} + b_{yor} + b_{rat} + b_{dev} + \alpha \cdot UserScore + \beta \cdot CriticScore + \gamma$$

This code fits the linear model:

```
###fits a linear model for critic and user score
fitFormula <- logGlobal_Sales -
  (log10_mu + t_v_b_pub + t_v_b_pla + t_v_b_ge +
    t_v_b_yor + t_v_b_rat + t_v_b_dev) ~ Critic_Score + User_Score
fit <- lm(fitFormula, data = scores_set )
summary(fit)
```

```
##
## Call:
## lm(formula = fitFormula, data = scores_set)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.09052 -0.25611  0.01421  0.26112  1.75392
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.5726299  0.0306142 -18.705   <2e-16 ***
## Critic_Score  0.0083741  0.0004705  17.797   <2e-16 ***
```

```
## User_Score   -0.0004389  0.0004530  -0.969    0.333
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4026 on 5704 degrees of freedom
## Multiple R-squared:  0.0727, Adjusted R-squared:  0.07237
## F-statistic: 223.6 on 2 and 5704 DF,  p-value: < 2.2e-16
```

```
intercept <- fit$coefficients[1]
alpha <- fit$coefficients[2]
beta <- fit$coefficients[3]
rm(fit)
```

From the R^2 of the fit, one can see that this model is far from perfect, but is a step further toward improving the previous model. Let's see if it lowers the RMSE:

```
###using the environmental bias and scores to make predictions
pred = 10^(log10_mu + vec_bias_pub + vec_bias_pla + vec_bias_ge +
             vec_bias_yor + vec_bias_rat + vec_bias_dev +
             intercept + alpha*test$Critic_Score + beta*test$User_Score)
RMSE <- fun_RMSE(test$Global_Sales, pred)
#keep track of RMSE
mat_RMSE <- rbind(mat_RMSE, c("LM: Worldwide with regularized environmental bias and score linear model"
```

```
##      Method
## [1,] "Worldwide naive approach: average"
## [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
## [3,] "LM: Worldwide with regularized environmental bias"
## [4,] "LM: Worldwide with regularized environmental bias and score linear model"
##      RMSE
## [1,] "3.83191547183535"
## [2,] "2.86489546914096"
## [3,] "2.60215666537728"
## [4,] "2.37764522461452"
```

The RMSE is now 2.38, which is again a slight improvement. The help of regularization should help to decrease it further:

```
###regularization of critic score
lambdas <- seq(20, 150, 1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_crit_score <- scores_set$sumCritScore/(lam + scores_set$Critic_Count) #regularized critic score
  pred <- 10^(log10_mu + t_v_b_pub + t_v_b_pla + t_v_b_ge +
          t_v_b_yor + t_v_b_rat + t_v_b_dev + intercept +
          alpha*reg_crit_score + beta*scores_set$User_Score)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
lambda_crit <- lambdas[which.min(RMSEs_train)]
#adding regularized column to test and train set
training <- training %>% mutate(Critic_Score_Reg = scores_set$sumCritScore/(lambda_crit + scores_set$Cr
test <- test %>% mutate(Critic_Score_Reg = (Critic_Score*Critic_Count)/(lambda_crit + Critic_Count))
```

```r
###regularization of user score
lambdas <- seq(-3, 2, 0.1)
#vector of RMSE for each lambda
RMSEs_train <- sapply(lambdas, function(lam){
  reg_user_score <- scores_set$sumUserScore/(lam + scores_set$User_Count) #regularized critic score
  pred <- 10^(log10_mu + t_v_b_pub + t_v_b_pla + t_v_b_ge +
                t_v_b_yor + t_v_b_rat + t_v_b_dev + intercept +
                alpha*training$Critic_Score_Reg + beta*reg_user_score)
  fun_RMSE(training$Global_Sales, pred)
})
#working out the lambda that minimizes training RMSE
lambda_user <- lambdas[which.min(RMSEs_train)]
#adding regularized column to test and train set
training <- training %>% mutate(User_Score_Reg = scores_set$sumUserScore/(lambda_user + scores_set$User_
test <- test %>% mutate(User_Score_Reg = (User_Score*User_Count)/(lambda_user + User_Count))

###computes the new RMSE on test set for regularized scores
pred = 10^(log10_mu + vec_bias_pub + vec_bias_pla + vec_bias_ge +
             vec_bias_yor + vec_bias_rat + vec_bias_dev + intercept +
             alpha*test$Critic_Score_Reg + beta*test$User_Score_Reg)
RMSE <- fun_RMSE(test$Global_Sales, pred)
#keep track of RMSE
mat_RMSE <- rbind(mat_RMSE, c("LM: Worldwide with environmental bias and score linear model both regula
```

```
##      Method
## [1,] "Worldwide naive approach: average"
## [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
## [3,] "LM: Worldwide with regularized environmental bias"
## [4,] "LM: Worldwide with regularized environmental bias and score linear model"
## [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
##      RMSE
## [1,] "3.83191547183535"
## [2,] "2.86489546914096"
## [3,] "2.60215666537728"
## [4,] "2.37764522461452"
## [5,] "0.849291081297686"
```

With the use of regularized reviews, the RMSE has a huge drop: it's now at 0.85. One can explain because there are lot of games with a good score (especially user score) and low review count.

The RMSE of 0.85 means that this model is somewhat accurate to predict video games sales: being less than 100% off the real sales number when looking at a market with such difference in scale is quite good.

II)C)3)Linear model to predict region sales

Let's see if the same model can spot regional trends: I can use it to predict the sales for a given game in each region (North America, European Union, Japan and the rest of the world), sum it to have a worldwide sales prediction and compare it to actual values.

Since it's the same process than before, I put it in a function. I first needed a bit of data processing: since I work with the log of sales and some games did not sale a single unit in some region, the log is $-\infty$, which makes it impossible to work with it. For this purpose, I arbitrarily chose that instead of selling no unit in a given region, the game sold just 1 unit. With that, the log is now 0 and everything is computable again.

```r
#use the fun_Region_Pred_LM function to predict the sales in each region
pred_NA <- fun_Region_Pred_LM(training$NA_Sales)
pred_EU <- fun_Region_Pred_LM(training$EU_Sales)
pred_JP <- fun_Region_Pred_LM(training$JP_Sales)
pred_Other <- fun_Region_Pred_LM(training$Other_Sales)



#########################################################
###      Region-wise: Final results
#########################################################

###computing worldwide final predictions
pred <- pred_NA + pred_EU + pred_JP + pred_Other
rm(pred_EU, pred_JP, pred_NA, pred_Other)
RMSE <- fun_RMSE(test$Global_Sales, pred)
###keeping track of the RMSE
mat_RMSE <- rbind(mat_RMSE, c("LM: Region-wise with environmental bias and score linear model both regul
```

```
##      Method
## [1,] "Worldwide naive approach: average"
## [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
## [3,] "LM: Worldwide with regularized environmental bias"
## [4,] "LM: Worldwide with regularized environmental bias and score linear model"
## [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
## [6,] "LM: Region-wise with environmental bias and score linear model both regularized"
##      RMSE
## [1,] "3.83191547183535"
## [2,] "2.86489546914096"
## [3,] "2.60215666537728"
## [4,] "2.37764522461452"
## [5,] "0.849291081297686"
## [6,] "0.920741129773057"
```

When using the predictions for each region, I end up with a RMSE of 0.92, which is slightly worse but very similar to the worldwide RMSE of 0.85. It means that this model can predict regional trends almost as well as worldwide trends. This model works well with different scales: for instance in Japan there are a few typical games that sold well and a lot of games saw very few sales there, but the model is still able to predict these few sales with a good accuracy.

II)D)Using K-nearest-neighbors algorithm

Since this project requires us to use at least two different algorithms, I chose to use the K-nearest-neighbors (KNN) algorithm. This algorithm searches for the K closest games (for a given distance) from the game whose sales are to be predicted, and predicts the mean of sales of these K games.

This algorithm has an advantage versus linear model: I had to guess what model I needed to use to predict the sales of a game (using the log of sales rather than sales themselves), with KNN I don't have to since it uses the nearest neighbor's value without having to fit a model.

II)D)1)Using KNN with review as a distance

The next step is to select a distance to work out the nearest neighbors of a game. The first thing I tried was the reviews, the only (apart from sales) numeric value of the dataset. This is is not really meant to have a great RMSE, but rather to find a good value of K.

```
###first estimation using only User and Critic score
train_knn <- train(Global_Sales ~ User_Score + Critic_Score,
             data = training,
             method = "knn",
             tuneGrid = data.frame(k = seq(3,15,2)))
train_knn$bestTune
```

```
##    k
## 7 15
```

```
pred <- predict(train_knn, test)
RMSE <- fun_RMSE(test$Global_Sales, pred)
mat_RMSE <- rbind(mat_RMSE, c("KNN: Worldwide with score review", RMSE))
```

It seems the best K to use is 15, which seemed a little high for me, since K is usually between 5 and 10. Furthermore, the metric use by KNN to pick the most accurate K is the absolute RMSE, not the relative one, which makes high selling games weight more than low selling games.

```
##      Method
## [1,] "Worldwide naive approach: average"
## [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
## [3,] "LM: Worldwide with regularized environmental bias"
## [4,] "LM: Worldwide with regularized environmental bias and score linear model"
## [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
## [6,] "LM: Region-wise with environmental bias and score linear model both regularized"
## [7,] "KNN: Worldwide with score review"
##      RMSE
## [1,] "3.83191547183535"
## [2,] "2.86489546914096"
## [3,] "2.60215666537728"
## [4,] "2.37764522461452"
## [5,] "0.849291081297686"
## [6,] "0.920741129773057"
## [7,] "9.89024242692307"
```

The result here is obviously not good, it is actually more than two times worse than the naive method of using the average, that is terribly bad. Let's see how much I can improve it by selecting another distance.

II)D)2)Using KNN with sum of sales environmental variables as a distance

I thought that using the sum of sales for each environmental variable was likely to provide a good estimate of the sales: I computed the sum of sales for each publisher, platform, genre, year of release, rating and developer. If two games are close for this distance, it means that they are likely to be of the same publisher, developer or genre, to be available on the same platform, etc. With this distance, two games that are neighbors are the one that share the most "environmental" variables. Let's construct this distance:

```
###creating a distance using sum of sales for each environmental variable
#compute the sum (and other data) for each publisher/platform/etc
pub <- training %>% group_by(Publisher) %>%
  summarise(sum = sum(Global_Sales),
            mean = mean(Global_Sales),
            count = n(),
            .groups="drop")
pla <- training %>% group_by(Platform) %>%
  summarise(sum = sum(Global_Sales),
            mean = mean(Global_Sales),
            count = n(),
            .groups="drop")
gen <- training %>% group_by(Genre) %>%
  summarise(sum = sum(Global_Sales),
            mean = mean(Global_Sales),
            count = n(),
            .groups="drop")
yor <- training %>% group_by(Year_of_Release) %>%
  summarise(sum = sum(Global_Sales),
            mean = mean(Global_Sales),
            count = n(),
            .groups="drop")
rat <- training %>% group_by(Rating) %>%
  summarise(sum = sum(Global_Sales),
            mean = mean(Global_Sales),
            count = n(),
            .groups="drop")
dev <- training %>% group_by(Developer) %>%
  summarise(sum = sum(Global_Sales),
            mean = mean(Global_Sales),
            count = n(),
            .groups="drop")

###creating the matching values to columns in training and test set
#each column added matches the value of the sum of sales for
#each possible value of publisher/platform/developer/etc
training$sum_Pub <- sapply(training$Publisher, function(x)
  pub$sum[which(pub$Publisher == x)])
training$sum_Pla <- sapply(training$Platform, function(x)
  pla$sum[which(pla$Platform == x)])
training$sum_Gen <- sapply(training$Genre, function(x)
  gen$sum[which(gen$Genre == x)])
training$sum_Yor <- sapply(training$Year_of_Release, function(x)
  yor$sum[which(yor$Year_of_Release == x)])
training$sum_Rat <- sapply(training$Publisher, function(x)
  pub$sum[which(pub$Publisher == x)])
training$sum_Dev <- sapply(training$Developer, function(x)
  dev$sum[which(dev$Developer == x)])
test$sum_Pub <- sapply(test$Publisher, function(x)
  pub$sum[which(pub$Publisher == x)])
test$sum_Pla <- sapply(test$Platform, function(x)
  pla$sum[which(pla$Platform == x)])
test$sum_Gen <- sapply(test$Genre, function(x)
```

```
  gen$sum[which(gen$Genre == x)])
test$sum_Yor <- sapply(test$Year_of_Release, function(x)
  yor$sum[which(yor$Year_of_Release == x)])
test$sum_Rat <- sapply(test$Publisher, function(x)
  pub$sum[which(pub$Publisher == x)])
test$sum_Dev <- sapply(test$Developer, function(x)
  dev$sum[which(dev$Developer == x)])
```

I also wanted to manually select the best K using cross-validation on the training set, which is done in this code:

```
###using cross validation to manually select the best K
set.seed(1, sample.kind = "Rounding")
Ks <- seq(3,15,2)
RMSEs_train <- sapply(Ks, function(x){
  #creates cross-validation sets
  indexes <- createDataPartition(training$gameId, times = 1, p = 0.2, list=FALSE)
  validation <- training[indexes,]
  inner_training <- training[-indexes,]
  #fits the model on the "inner training" set
  train_knn <- train(Global_Sales ~ sum_Pub+sum_Pla+sum_Gen+sum_Yor+sum_Rat+sum_Dev,
                     data = inner_training,
                     method = "knn",
                     tuneGrid = data.frame(k = x))
  pred <- predict(train_knn, validation)
  fun_RMSE(validation$Global_Sales, pred)
})
optiK <- Ks[which.min(RMSEs_train)]
rm(RMSEs_train, Ks)
optiK
```

```
## [1] 5
```

It turns out the optimal value of K for the RMSE I'm using is likely to be 5. Let's use KNN on the sum of sales for environmental variables for K=5:

```
###knn using the sum for each environmental variable
train_knn <- train(Global_Sales ~ sum_Pub+sum_Pla+sum_Gen+sum_Yor+sum_Rat+sum_Dev,
                   data = training,
                   method = "knn",
                   tuneGrid = data.frame(k = optiK))
pred <- predict(train_knn, test)
RMSE <- fun_RMSE(test$Global_Sales, pred)
mat_RMSE <- rbind(mat_RMSE, c("KNN with sum of sales for environmental variables, K from cross validati
```

```
##      Method
## [1,] "Worldwide naive approach: average"
## [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
## [3,] "LM: Worldwide with regularized environmental bias"
## [4,] "LM: Worldwide with regularized environmental bias and score linear model"
## [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
## [6,] "LM: Region-wise with environmental bias and score linear model both regularized"
```

```
## [7,] "KNN: Worldwide with score review"
## [8,] "KNN with sum of sales for environmental variables, K from cross validation"
##      RMSE
## [1,] "3.83191547183535"
## [2,] "2.86489546914096"
## [3,] "2.60215666537728"
## [4,] "2.37764522461452"
## [5,] "0.849291081297686"
## [6,] "0.920741129773057"
## [7,] "9.89024242692307"
## [8,] "7.19958191675274"
```

The RMSE is now 7.2, which is better but still way worse than any RMSE I got using the linear model.

II)D)3)Using KNN with mean of sales environmental variables as a distance

I tried a version with the mean (instead of the sum) of sales for each environmental variable, which improves the RMSE a little bit:

```
###removing columns for sum of sales
training <- training %>% select(-sum_Pub, -sum_Pla, -sum_Gen, -sum_Yor, -sum_Rat, -sum_Dev)
test <- test %>% select(-sum_Pub, -sum_Pla, -sum_Gen, -sum_Yor, -sum_Rat, -sum_Dev)

###creating the matching values to columns in training and test set
#each column added matches the value of the mean of sales for
#each possible value of publisher/platform/developer/etc
training$mean_Pub <- sapply(training$Publisher, function(x)
  pub$mean[which(pub$Publisher == x)])
training$mean_Pla <- sapply(training$Platform, function(x)
  pla$mean[which(pla$Platform == x)])
training$mean_Gen <- sapply(training$Genre, function(x)
  gen$mean[which(gen$Genre == x)])
training$mean_Yor <- sapply(training$Year_of_Release, function(x)
  yor$mean[which(yor$Year_of_Release == x)])
training$mean_Rat <- sapply(training$Publisher, function(x)
  pub$mean[which(pub$Publisher == x)])
training$mean_Dev <- sapply(training$Developer, function(x)
  dev$mean[which(dev$Developer == x)])
test$mean_Pub <- sapply(test$Publisher, function(x)
  pub$mean[which(pub$Publisher == x)])
test$mean_Pla <- sapply(test$Platform, function(x)
  pla$mean[which(pla$Platform == x)])
test$mean_Gen <- sapply(test$Genre, function(x)
  gen$mean[which(gen$Genre == x)])
test$mean_Yor <- sapply(test$Year_of_Release, function(x)
  yor$mean[which(yor$Year_of_Release == x)])
test$mean_Rat <- sapply(test$Publisher, function(x)
  pub$mean[which(pub$Publisher == x)])
test$mean_Dev <- sapply(test$Developer, function(x)
  dev$mean[which(dev$Developer == x)])

###using cross validation to manually select the best K
set.seed(42, sample.kind = "Rounding")
```

```r
Ks <- seq(3,15,2)
RMSEs_train <- sapply(Ks, function(x){
  #creates cross-validation sets
  indexes <- createDataPartition(training$gameId, times = 1, p = 0.2, list=FALSE)
  validation <- training[indexes,]
  inner_training <- training[-indexes,]
  train_knn <- train(Global_Sales ~ mean_Pub+mean_Pla+mean_Gen+mean_Yor+mean_Rat+mean_Dev,
                     data = inner_training,
                     method = "knn",
                     tuneGrid = data.frame(k = x))
  pred <- predict(train_knn, validation)
  fun_RMSE(validation$Global_Sales, pred)
})
optiK <- Ks[which.min(RMSEs_train)]
rm(RMSEs_train, Ks)

###using the mean for each environmental variable
train_knn <- train(Global_Sales ~ mean_Pub+mean_Pla+mean_Gen+mean_Yor+mean_Rat+mean_Dev,
                   data = training,
                   method = "knn",
                   tuneGrid = data.frame(k = optiK))
pred<- predict(train_knn, test)
RMSE <- fun_RMSE(test$Global_Sales, pred)
mat_RMSE <- rbind(mat_RMSE, c("KNN: Worldwide with mean of sales for environmental variables, K from cr

###free memory
#delete variables
rm(dev, gen, pla, pub, rat, yor, train_knn)
#removing columns for mean of sales
training <- training %>% select(-mean_Pub, -mean_Pla, -mean_Gen, -mean_Yor, -mean_Rat, -mean_Dev)
test <- test %>% select(-mean_Pub, -mean_Pla, -mean_Gen, -mean_Yor, -mean_Rat, -mean_Dev)
```

```
##       Method
##  [1,] "Worldwide naive approach: average"
##  [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
##  [3,] "LM: Worldwide with regularized environmental bias"
##  [4,] "LM: Worldwide with regularized environmental bias and score linear model"
##  [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
##  [6,] "LM: Region-wise with environmental bias and score linear model both regularized"
##  [7,] "KNN: Worldwide with score review"
##  [8,] "KNN with sum of sales for environmental variables, K from cross validation"
##  [9,] "KNN: Worldwide with mean of sales for environmental variables, K from cross validation"
##       RMSE
##  [1,] "3.83191547183535"
##  [2,] "2.86489546914096"
##  [3,] "2.60215666537728"
##  [4,] "2.37764522461452"
##  [5,] "0.849291081297686"
##  [6,] "0.920741129773057"
##  [7,] "9.89024242692307"
##  [8,] "7.19958191675274"
##  [9,] "7.12081885708865"
```

II)D)4)KNN to predict region sales

Just like the linear model, I made a function, that is essentially a copy/paste of the previous code, to try to predict region sales. The aim is to see if KNN can spot regional trends in sales.

```
###use the fun_Region_Pred_KNN function to predict the sales in each region
pred_NA <- fun_Region_Pred_KNN(training$NA_Sales)
pred_EU <- fun_Region_Pred_KNN(training$EU_Sales)
pred_JP <- fun_Region_Pred_KNN(training$JP_Sales)
pred_Other <- fun_Region_Pred_KNN(training$Other_Sales)

###computing worldwide final predictions
pred <- pred_NA + pred_EU + pred_JP + pred_Other
rm(pred_EU, pred_JP, pred_NA, pred_Other)
RMSE <- fun_RMSE(test$Global_Sales, pred)
###keeping track of the RMSE
mat_RMSE <- rbind(mat_RMSE, c("KNN: Region-wise with mean of sales for environmental variables, K from
```

```
##       Method
##  [1,] "Worldwide naive approach: average"
##  [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
##  [3,] "LM: Worldwide with regularized environmental bias"
##  [4,] "LM: Worldwide with regularized environmental bias and score linear model"
##  [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
##  [6,] "LM: Region-wise with environmental bias and score linear model both regularized"
##  [7,] "KNN: Worldwide with score review"
##  [8,] "KNN with sum of sales for environmental variables, K from cross validation"
##  [9,] "KNN: Worldwide with mean of sales for environmental variables, K from cross validation"
## [10,] "KNN: Region-wise with mean of sales for environmental variables, K from cross validation"
##       RMSE
##  [1,] "3.83191547183535"
##  [2,] "2.86489546914096"
##  [3,] "2.60215666537728"
##  [4,] "2.37764522461452"
##  [5,] "0.849291081297686"
##  [6,] "0.920741129773057"
##  [7,] "9.89024242692307"
##  [8,] "7.19958191675274"
##  [9,] "7.12081885708865"
## [10,] "25.3433507470235"
```

The RMSE is now greater than 25. Just to recall, it means the prediction is 2500% away from the real sales, that is around 6 times worse than the naive prediction! With this, I know that the KNN method is unable to detect small trends. Which made me think that KNN may actually be unable to do good predictions for small values of sales. It requires further exploration.

II)D)4)Exploration of KNN predictions

I created a training subset, to use sort of a cross-validation (with only one resample).

```
###creates sets for cross validation
set.seed(1234, sample.kind = "Rounding")
```

```
#20% of validation, 80% of "inner training"
indexes <- createDataPartition(training$gameId, times = 1, p = 0.2, list=FALSE)
sub_training <- training[-indexes,]
temp <- training[indexes,]
validation <- temp %>%
  semi_join(sub_training, by = "Platform") %>%
  semi_join(sub_training, by = "Genre") %>%
  semi_join(sub_training, by = "Year_of_Release") %>%
  semi_join(sub_training, by = "Publisher") %>%
  semi_join(sub_training, by = "Rating") %>%
  semi_join(sub_training, by = "Developer")
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("gameId", "Name", "Platform", "Year_of_Release", "Genre", "Publisher", "NA_Sales", "
```

```
sub_training <- rbind(sub_training, removed)
rm(temp, removed)
```

I then had some trouble using my previous functions to make predictions for both linear model and K-nearest-neighbors, so I ran the entire process again, the code is not included here because it's very long and similar to what I've already done. pred_LM and pred_KNN are the predictions on the validation set respectively for the linear model and for K-nearest-neighbors.

Once the predictions for the validation set are done, I computed the squared error for each prediction:

```
###compute the squared error and the RMSE for both KNN and LM
fun_RMSE(validation$Global_Sales, pred_LM)
```

```
## [1] 1.685038
```

```
fun_RMSE(validation$Global_Sales, pred_KNN)
```

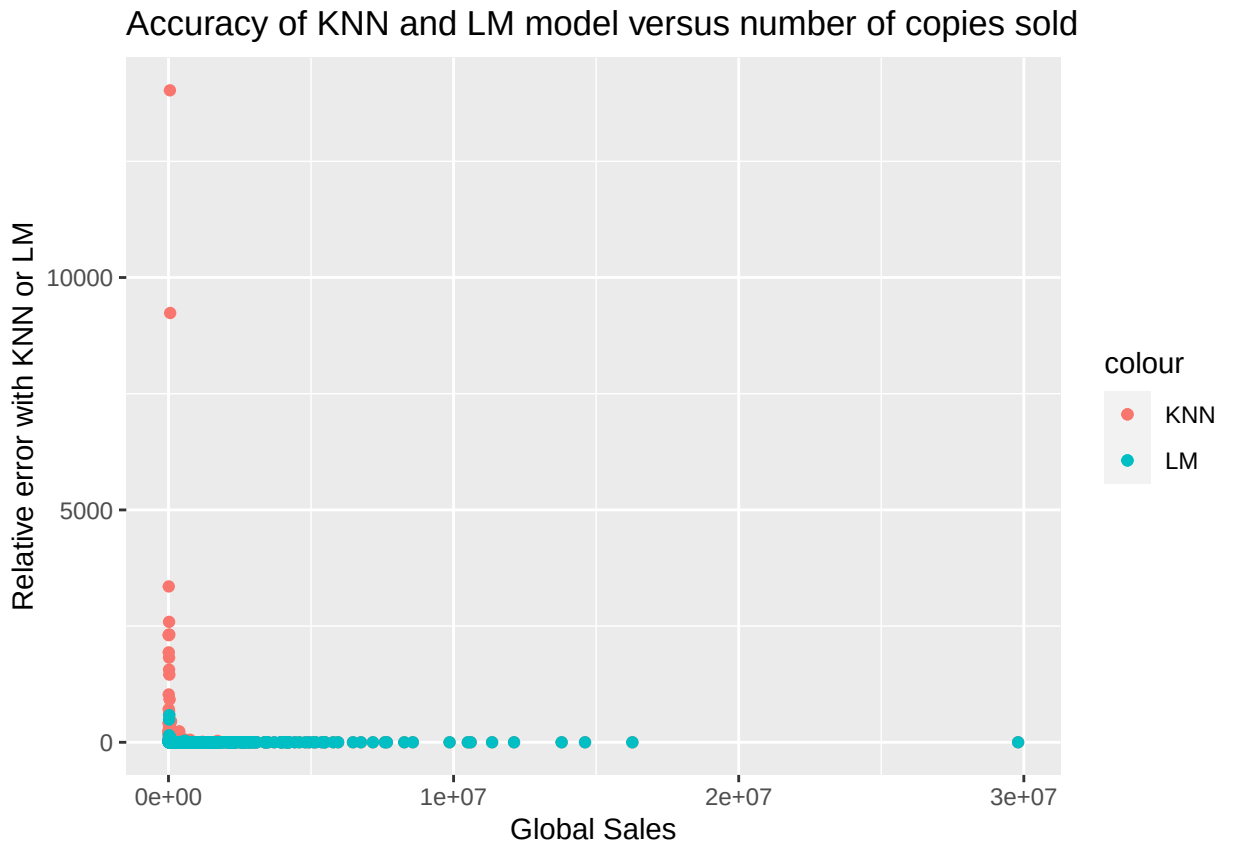
```
## [1] 7.825619
```

```
SE_rel_LM <- ( (validation$Global_Sales - pred_LM)/validation$Global_Sales )^2
SE_rel_KNN <- ( (validation$Global_Sales - pred_KNN)/validation$Global_Sales )^2

###EDA
mean(SE_rel_KNN < SE_rel_LM)
```

```
## [1] 0.4903943
```

```
df <- data.frame(validation$Global_Sales, SE_rel_LM, SE_rel_KNN)
```

Just like before, the RMSE is much higher for KNN than for LM, but I was surprised to see that almost half the predictions are better with the KNN algorithm. It means that KNN is slightly better than LM half of the time, and way worse the other half of the time... but is there som kind of trend?



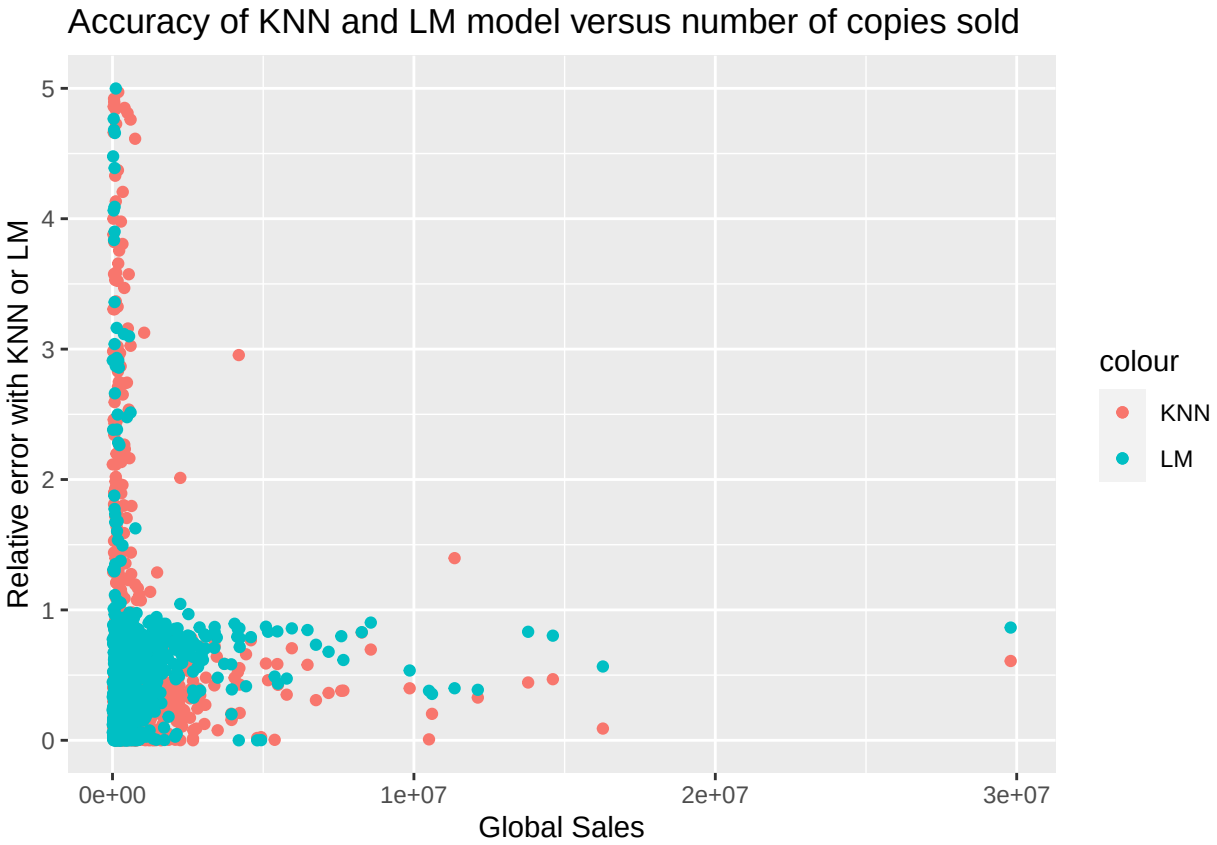
On this first graph we can see that there are some very high errors for KNN, these errors drive the RMSE towards higher values, but they are mainly made for games with low amount of sales. We can reduce the scope of the graphic: when considering only predictions with an squared error lesser than 5, we still have about 94% of the LM predictions, and 78% of the KNN predictions.

```
mean(SE_rel_KNN < 5)
```

```
## [1] 0.776542
```

```
mean(SE_rel_LM < 5)
```

```
## [1] 0.9443883
```



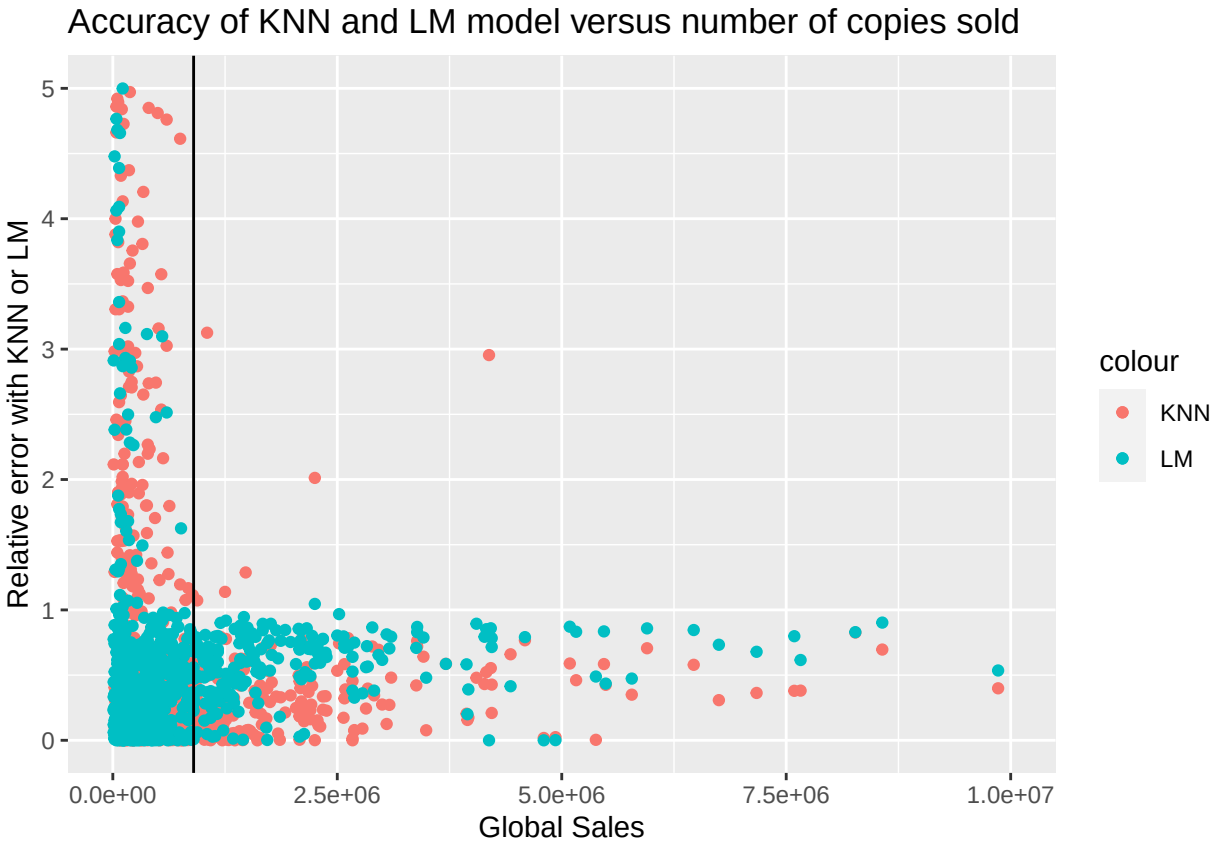
With this second graph, we can start to spot a trend: above a given amount of sales, KNN predictions tend to be more accurate than LM. Let's reduce the scope again: when considering only predictions with an squared error lesser than 1, we still have about 89% of the LM predictions, and 64% of the KNN predictions.

```
mean(SE_rel_KNN < 1)
```

```
## [1] 0.6380182
```

```
mean(SE_rel_LM < 1)
```

```
## [1] 0.8948433
```

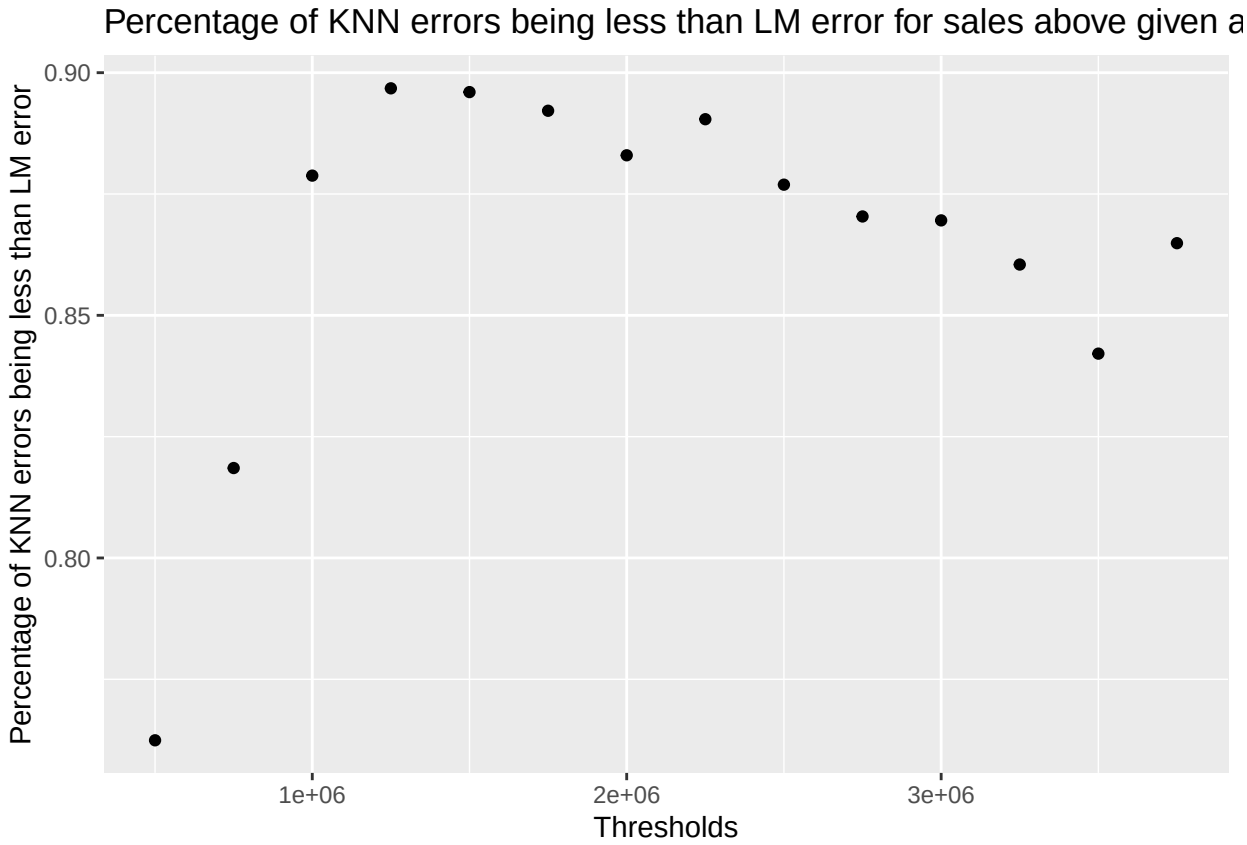


On this last graph we can try to estimate the threshold above which KNN is likely to be more accurate than LM. Above is a graphic representation of such a threshold: on the right hand side of the threshold, almost all the red dots (KNN predictions) are below (more accurate) the blue dots (LM predictions). With the threshold above (0.09×10^7), 88% percent of KNN predictions after said threshold are more accurate than LM predictions.

```
threshold <- 0.09*10^7
mean(SE_rel_KNN[which(validation$Global_Sales > threshold)] <
       SE_rel_LM[which(validation$Global_Sales > threshold)])
```

```
## [1] 0.8798077
```

However, it only applies to a small amount of predictions. We can compute a better estimate of the ideal



threshold:

```
max(means)
```

```
## [1] 0.8967742
```

```
threshold <- threshs[which.max(means)]
threshold
```

```
## [1] 1250000
```

The ideal threshold on this training subset is 0.125×10^7, achieving a better result with KNN for almost 90% of the predictions. Let's see if we can build another model using this threshold.

II)E)Building a hybrid theory

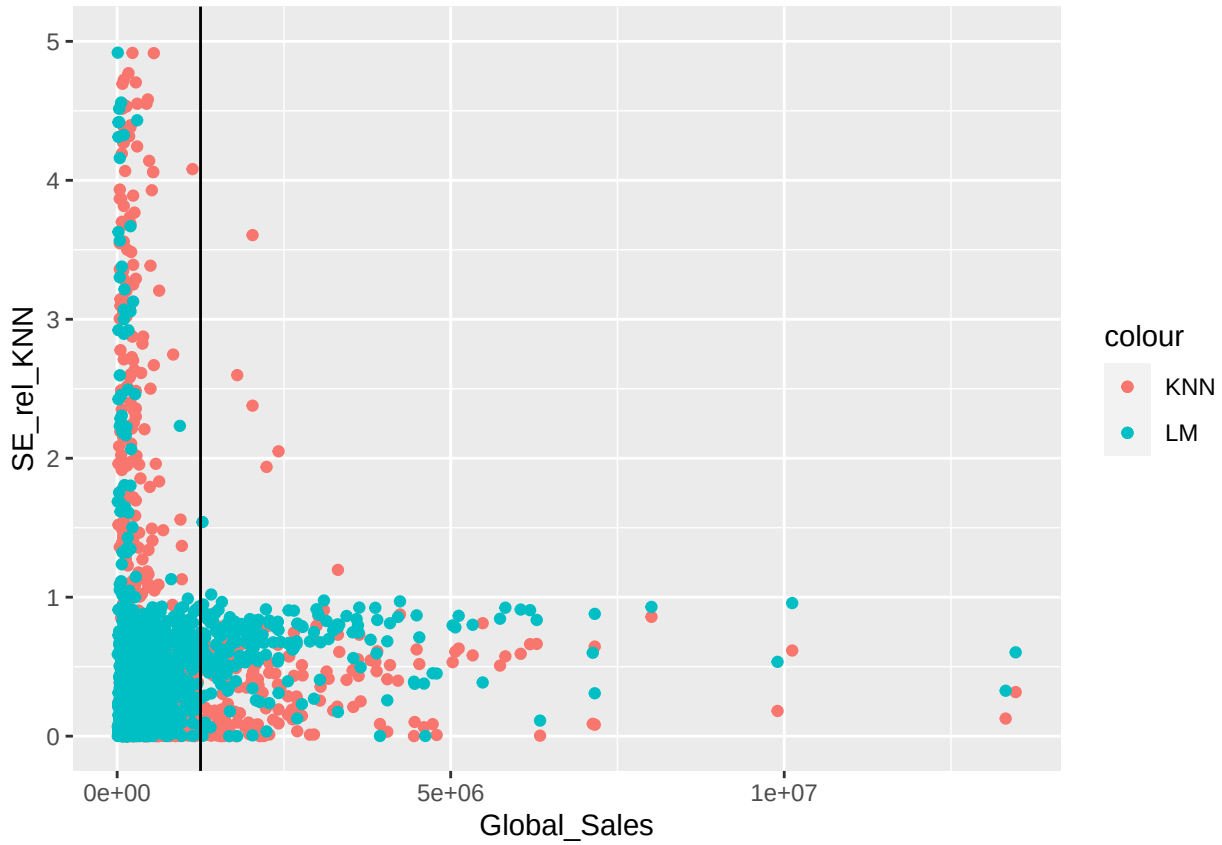
With all the models previously made, I can summarize the observations: a linear model has a bad accuracy for low sales but a good accuracy for games with a high amount of sales; whereas K-nearest-neighbors algorithm has terrible accuracy for low sales and a very good accuracy for high-selling games. One would want the best of both models.

This is actually quite easy to make: it requires to make predictions for bith KNN and LM, and for each game if a given test is passed, we select the KNN prediction, otherwise we select the LM prediction. The said test should involve LM and/or KNN predictions and the threshold computed earlier on the training subset. I chose to use "if the linear model prediction is higher than the threshold" as a test, because since the KNN predictions have more variability, using only the linear model prediction gives more reliability to this hybrid theory. Here is the code to implement it:

```r
###making prediction using hybrid theory
pred_LM <- fun_Region_Pred_LM(training$Global_Sales)
pred_KNN <- fun_Region_Pred_KNN(training$Global_Sales)

###combining the predictions
pred <- ifelse( pred_LM>threshold, pred_KNN, pred_LM)
RMSE <- fun_RMSE(test$Global_Sales, pred)
mat_RMSE <- rbind(mat_RMSE, c("Hybrid theory (LM under a given threshold, KNN above said threshold): Wo
```

```
##       Method
##  [1,] "Worldwide naive approach: average"
##  [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
##  [3,] "LM: Worldwide with regularized environmental bias"
##  [4,] "LM: Worldwide with regularized environmental bias and score linear model"
##  [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
##  [6,] "LM: Region-wise with environmental bias and score linear model both regularized"
##  [7,] "KNN: Worldwide with score review"
##  [8,] "KNN with sum of sales for environmental variables, K from cross validation"
##  [9,] "KNN: Worldwide with mean of sales for environmental variables, K from cross validation"
## [10,] "KNN: Region-wise with mean of sales for environmental variables, K from cross validation"
## [11,] "Hybrid theory (LM under a given threshold, KNN above said threshold): Worldwide"
##       RMSE
##  [1,] "3.83191547183535"
##  [2,] "2.86489546914096"
##  [3,] "2.60215666537728"
##  [4,] "2.37764522461452"
##  [5,] "0.849291081297686"
##  [6,] "0.920741129773057"
##  [7,] "9.89024242692307"
##  [8,] "7.19958191675274"
##  [9,] "7.12081885708865"
## [10,] "25.3433507470235"
## [11,] "2.05148476155106"
```

The RMSE of this model is 2.05, which is both quite good and a little disappointing. It's quite good because it's by far the best RMSE I have for a model including KNN (it's even better than most linear models' RMSE), but it is still disappointing because I thought the use of KNN would decrease the RMSE even further than the best linear model. To understand why it didn't do so, we can look at the test data (that are similar to the validation subset shown earlier):



As I planned, most of the predictions above the threshold are better for KNN, but in the end it only includes a minority of games. Furthermore, the improvement of LM over KNN for low sales is greater than the improvement of KNN over LM for high sales, and since there are more games with low sales than high sales, the linear model stays the best model overall. One can try to push the threshold further, so that there are more LM predictions, but the model will then asymptotically be equivalent to the linear model, with very little improvement for high sales.

When using the relative RMSE as metric, this model is not better than the linear model since every game is worth the same and this model is only better for a few percents of the top sales. However, if I were to use the absolute RMSE as a metric, this model would likely be better than the linear model, since it makes the same errors for low sales (not very impactful for absolute RMSE) and that the slight percentage improvement for top sales would become a huge absolute improvement.

III)Results

III)A)RMSE imporvement

Here is the final RMSE table:

```
##       Method
##  [1,] "Worldwide naive approach: average"
##  [2,] "LM: Worldwide with environmental bias (publisher, platform, genre, YoR, rating and dev)"
##  [3,] "LM: Worldwide with regularized environmental bias"
##  [4,] "LM: Worldwide with regularized environmental bias and score linear model"
##  [5,] "LM: Worldwide with environmental bias and score linear model both regularized"
```

```
##  [6,] "LM: Region-wise with environmental bias and score linear model both regularized"
##  [7,] "KNN: Worldwide with score review"
##  [8,] "KNN with sum of sales for environmental variables, K from cross validation"
##  [9,] "KNN: Worldwide with mean of sales for environmental variables, K from cross validation"
## [10,] "KNN: Region-wise with mean of sales for environmental variables, K from cross validation"
## [11,] "Hybrid theory (LM under a given threshold, KNN above said threshold): Worldwide"
##       RMSE
##  [1,] "3.83191547183535"
##  [2,] "2.86489546914096"
##  [3,] "2.60215666537728"
##  [4,] "2.37764522461452"
##  [5,] "0.849291081297686"
##  [6,] "0.920741129773057"
##  [7,] "9.89024242692307"
##  [8,] "7.19958191675274"
##  [9,] "7.12081885708865"
## [10,] "25.3433507470235"
## [11,] "2.05148476155106"

## [1] "Best model:"

##                                                                                 Method
## "LM: Worldwide with environmental bias and score linear model both regularized"
##                                                                                   RMSE
##                                                                  "0.849291081297686"
```

The best model is a linear model, for which predictions are given by:

$$log_{10}(prediction) = log_{10}(\mu) + b_{pub} + b_{pla} + b_{ge} + b_{yor} + b_{rat} + b_{dev} + \alpha \cdot UserScore + \beta \cdot CriticScore + \gamma$$

With μ being the average amount of copies sold, b the bias, respectively for Publisher, Platform, Genre, Year of release, Rating and Developer, and α β and γ being constants that optimize the linear model predictions. All the biases and scores used in this model are regularized.

III)B)Discussion

We can see that RMSE has a high variability for different methods. The best RMSE I computed is below 100%, meanwhile the worst is above 2500%. Even though this ration of 25 is very impressive, that doesn't mean the best model here is objectively better under every circumstance. There are at least two reasons to be acknowledged:

- The result discussed here is obviously dependent from the metric of evaluation I chose earlier: KNN algorithm has a worse RMSE, but has better predictions for high selling games, which means the absolute RMSE for KNN could be better than the one for linear model.
- The video games market is particular in the way that there are huge differences of scale between high selling AAA games and small independent games. This particular configuration may make KNN less effective in it's predictions, because even the nearest-neighbors (when it comes to environmental values, i.e. publisher, platform, genre, release date, rating and developer) may already be on a different scale when it comes to sales.

Furthermore, the linear model is not entirely linear, since the predictions do not have a linear relationship with predictors, but instead the log of predictions have a linear relationship with the log of predictors, which accounts for multiplicative effects. KNN does not allow that.

To sum things up, the linear model better suited this particular dataset, due to the error metric used, scale effects and the use of multiplicative (rather than additive) biases.

IV)Conclusion

IV)A)Summary

I studied the the Video Games Sales dataset available on Kaggle. The aim was to predict the worldwide sales for games given their environmental variables (publisher, platform, genre, release date, ESRB rating and developer) and their Metacritic review score. This aim seems light-hearted or irrelevant, but the video games industry is already a huge economic sector, that is still growing to this day. The relative RMSE was used as a metric of evaluation. The two main models used were a linear model (with regularization) and the K-nearest-neighbors algorithm (with cross-validation). I briefly tried to combine both models to get the best of both.

The best model, a linear model, achieved a RMSE of 85%, which seems a lot at the first glance, but is actually not that bad given the scale difference in sales specific to the video game industry. However, one must recall that this is foremost an educational project (I had no previous experience in data science), so the results here may be greatly improvable for any skilled data scientist out there.

IV)B)Limitation and possible improvements

Limitation for this project mainly come from my own skills being limited: I chose to use K-nearest-neighbors because I was not sure how to properly use more advanced techniques (such as principle component analysis or matrix factorization) for this project.

Hardware improvement are not to be very useful here, because the dataset is relatively small (1.54 MB) so that even most complex algorithms will be quite fast to run.

Due to said skill limitations, I am not really sure how I could significantly improve the project myself. One of the only thing I thought was using a version of KNN that would compute the geometric mean (instead of the arithmetic mean) of the nearest neighbors' sales to improve KNN predictions, especially for low selling games (the geometric mean is lesser than the arithmetic one, whereas KNN predictions tend to be higher than the actual values). However, I do not doubt someone more skilled than me could improve this model further.

Thanks for reading, have a great day!