VideoGamesSales

Lukasz Kolodziejek

03/10/2019

Introduction

Overview

This is VideoGameSales Capstone project 2 for online course HarvardX: PH125.9x

Capstone project 2 gives opportunity to use a publicly available dataset to solve the problem of own choice.

Goal is to demonstrate skills important for data scientists, which includes clear communication of the process and insights gained from an analysis and proper documentation.

In this project Video Games Sales with ratings dataset from Kaggle will be used.

Dataset can be downloaded from here.

As access to file on Kaggle requires authorization for purpose of this project dataset copy has been created on Author's GitHub to allow for automatic download from R code:

https://github.com/lkolodziejek/VideoGameSales/blob/master/Video_Games_Sales_as_at_22_Dec_2016.csv

In scope of this project Video Games Sales data will be analysed and multiple models will be used to try to predict Global Sales with highest accuracy. Mean Root Square Error (MRSE) will be used to measure performance of models.

After initial data anslysis certain Features will be selected to be used in the models.

Analysis

Libraries and initial settings

First required libraries are being installed and loaded:

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(hexbin)) install.packages("hexbin", repos = "http://cran.us.r-project.org")
if(!require(doParallel)) install.packages("doParallel", repos = "http://cran.us.r-project.org")
```

Next, as calculations in this project requires a lot of computing power, parallel processing is enabled to leverage all CPUs and cores:

```
split <- detectCores()
cl <- makePSOCKcluster(split)
registerDoParallel(cl)
```

Data ingestion

Data is being downloaded from author's GitHub account and loaded. GitHub is being used for automatic data downloading, as Kaggle requires authorization and copy of data might not always be available.

```
dl <- tempfile()
download.file("https://github.com/lkolodziejek/VideoGameSales/blob/master/Video_Games_Sales_as_at_22_De
games <- fread("Video_Games_Sales_as_at_22_Dec_2016.csv", header=TRUE)
```

Data wrangling

After loading data is being wrangled and cleaned.

Empty fields are being replaced with NA, numeric values are being formatted as numbers. Categorical data are being formatted as factors and finally only data from before 2017 is being kept, as more recent data is incomplete.

```
games[games==""] <- NA #Fill missing values with NA
games$Year_of_Release <- as.numeric(as.character(games$Year_of_Release))
games$User_Score <- as.numeric(as.character(games$User_Score))*10 #Normalize User_Score to same scale a
games$Genre <- as.factor(games$Genre)
games$Publisher <- as.factor(games$Publisher)
games$Developer <- as.factor(games$Developer)
games$Rating <- as.factor(games$Rating)
games$Platform <- as.factor(games$Platform)
games <- games[-which(games$Year_of_Release>2016)] #As data after 2016 is not complete, remove it
```

‘‘‘

Data exploration

Our data analysis starts with description of Columns and basic statistics about them:

```
summary(games)
```

```
##      Name              Platform    Year_of_Release          Genre
##  Length:16715       PS2    :2161   Min.   :1980    Action      :3369
##  Class :character   DS     :2151   1st Qu.:2003    Sports      :2348
##  Mode  :character   PS3    :1331   Median :2007    Misc        :1750
##                     Wii    :1320   Mean   :2006    Role-Playing:1498
##                     X360   :1262   3rd Qu.:2010    Shooter     :1323
##                     PSP    :1209   Max.   :2016    (Other)     :6425
##                     (Other):7281   NA's   :269     NA's        :   2
##                         Publisher        NA_Sales          EU_Sales
##  Electronic Arts             : 1356   Min.   : 0.0000   Min.   : 0.0000
##  Activision                  :  985   1st Qu.: 0.0000   1st Qu.: 0.0000
##  Namco Bandai Games          :  939   Median : 0.0800   Median : 0.0200
##  Ubisoft                     :  932   Mean   : 0.2634   Mean   : 0.1451
##  Konami Digital Entertainment:  834   3rd Qu.: 0.2400   3rd Qu.: 0.1100
##  THQ                         :  715   Max.   :41.3600   Max.   :28.9600
##  (Other)                     :10954
```

```
##     JP_Sales          Other_Sales        Global_Sales      Critic_Score
##  Min.   : 0.00000   Min.   : 0.00000   Min.   : 0.0100   Min.   :13.00
##  1st Qu.: 0.00000   1st Qu.: 0.00000   1st Qu.: 0.0600   1st Qu.:60.00
##  Median : 0.00000   Median : 0.01000   Median : 0.1700   Median :71.00
##  Mean   : 0.07762   Mean   : 0.04734   Mean   : 0.5336   Mean   :68.97
##  3rd Qu.: 0.04000   3rd Qu.: 0.03000   3rd Qu.: 0.4700   3rd Qu.:79.00
##  Max.   :10.22000   Max.   :10.57000   Max.   :82.5300   Max.   :98.00
##                                                          NA's   :8578
##   Critic_Count      User_Score      User_Count          Developer
##  Min.   :  3.00   Min.   : 0.00   Min.   :    4.0   Ubisoft  : 203
##  1st Qu.: 12.00   1st Qu.:64.00   1st Qu.:   10.0   EA Sports: 172
##  Median : 21.00   Median :75.00   Median :   24.0   EA Canada: 167
##  Mean   : 26.36   Mean   :71.25   Mean   :  162.2   Konami   : 162
##  3rd Qu.: 36.00   3rd Qu.:82.00   3rd Qu.:   81.0   Capcom   : 139
##  Max.   :113.00   Max.   :97.00   Max.   :10665.0   (Other)  :9252
##  NA's   :8578     NA's   :9125    NA's   :9125      NA's     :6620
##      Rating
##  E      :3990
##  T      :2961
##  M      :1563
##  E10+   :1420
##  EC     :   8
##  (Other):   7
##  NA's   :6766
```

We also check dimensions of the data:

```
dim(games)
```

```
## [1] 16715    16
```

And list top rows to better understand data structure:

```
head(games)
```

```
##                       Name Platform Year_of_Release        Genre
## 1:                Wii Sports      Wii            2006       Sports
## 2:         Super Mario Bros.      NES            1985     Platform
## 3:            Mario Kart Wii      Wii            2008       Racing
## 4:         Wii Sports Resort      Wii            2009       Sports
## 5: Pokemon Red/Pokemon Blue       GB            1996 Role-Playing
## 6:                   Tetris       GB            1989       Puzzle
##    Publisher NA_Sales EU_Sales JP_Sales Other_Sales Global_Sales
## 1:  Nintendo    41.36    28.96     3.77        8.45        82.53
## 2:  Nintendo    29.08     3.58     6.81        0.77        40.24
## 3:  Nintendo    15.68    12.76     3.79        3.29        35.52
## 4:  Nintendo    15.61    10.93     3.28        2.95        32.77
## 5:  Nintendo    11.27     8.89    10.22        1.00        31.37
## 6:  Nintendo    23.20     2.26     4.22        0.58        30.26
##    Critic_Score Critic_Count User_Score User_Count Developer Rating
## 1:           76           51         80        322  Nintendo      E
## 2:           NA           NA         NA         NA      <NA>   <NA>
```

```
## 3:           82           73         83        709  Nintendo      E
## 4:           80           73         80        192  Nintendo      E
## 5:           NA           NA         NA         NA      <NA>   <NA>
## 6:           NA           NA         NA         NA      <NA>   <NA>
```

As we can see dataset contains information about games sales and provides data like:

- Game Name
- Platform on which it has been released
- Year of release
- Game genre
- Publisher
- Sales per region and it's sum (Global Sales)
- Critic Score and numer of critics reviews
- User Score and number of users reviews
- Developer

Unfortunately some of the data are missing and there we can see NA value.

We can display number of games released per year (if game has been released on 3 different platforms, its being counted as 3):

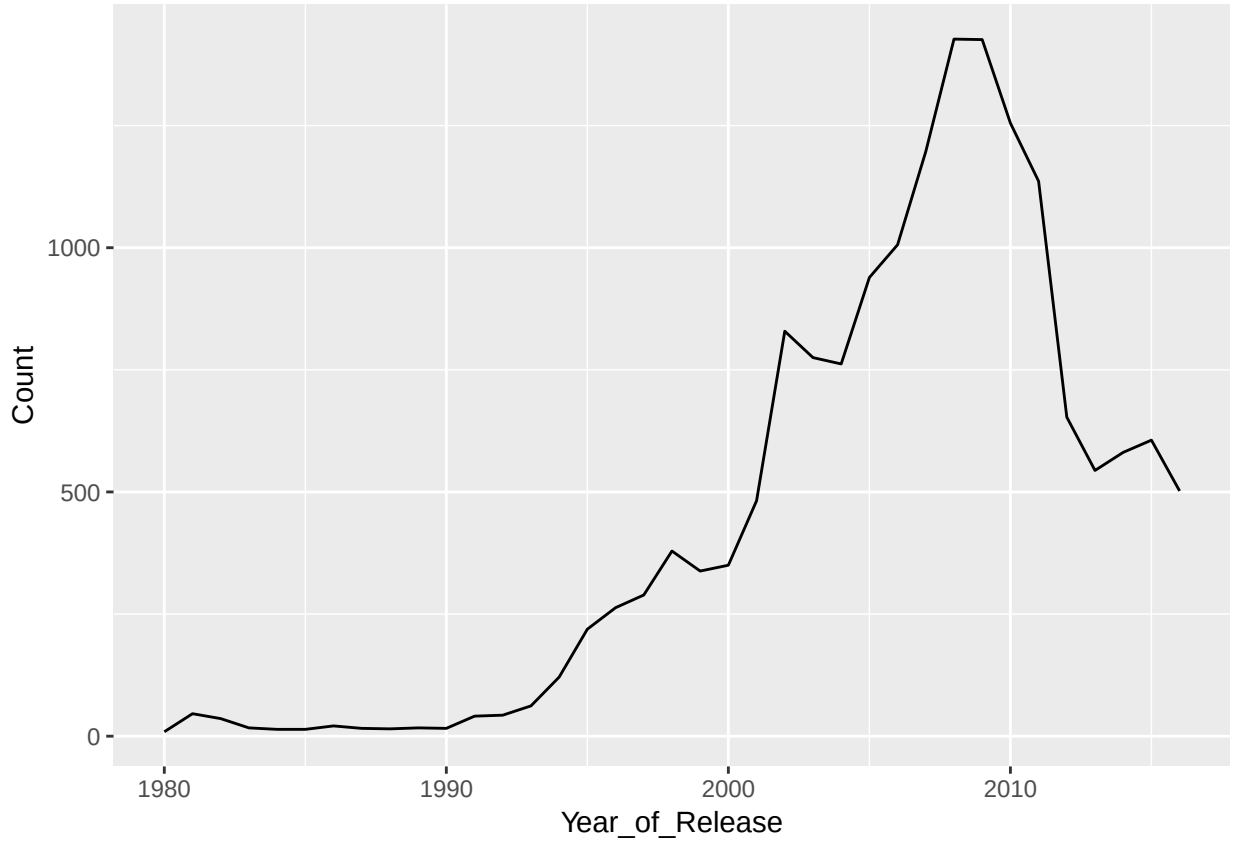

Figure 1: Number of games released per year

Games sales per region in time:

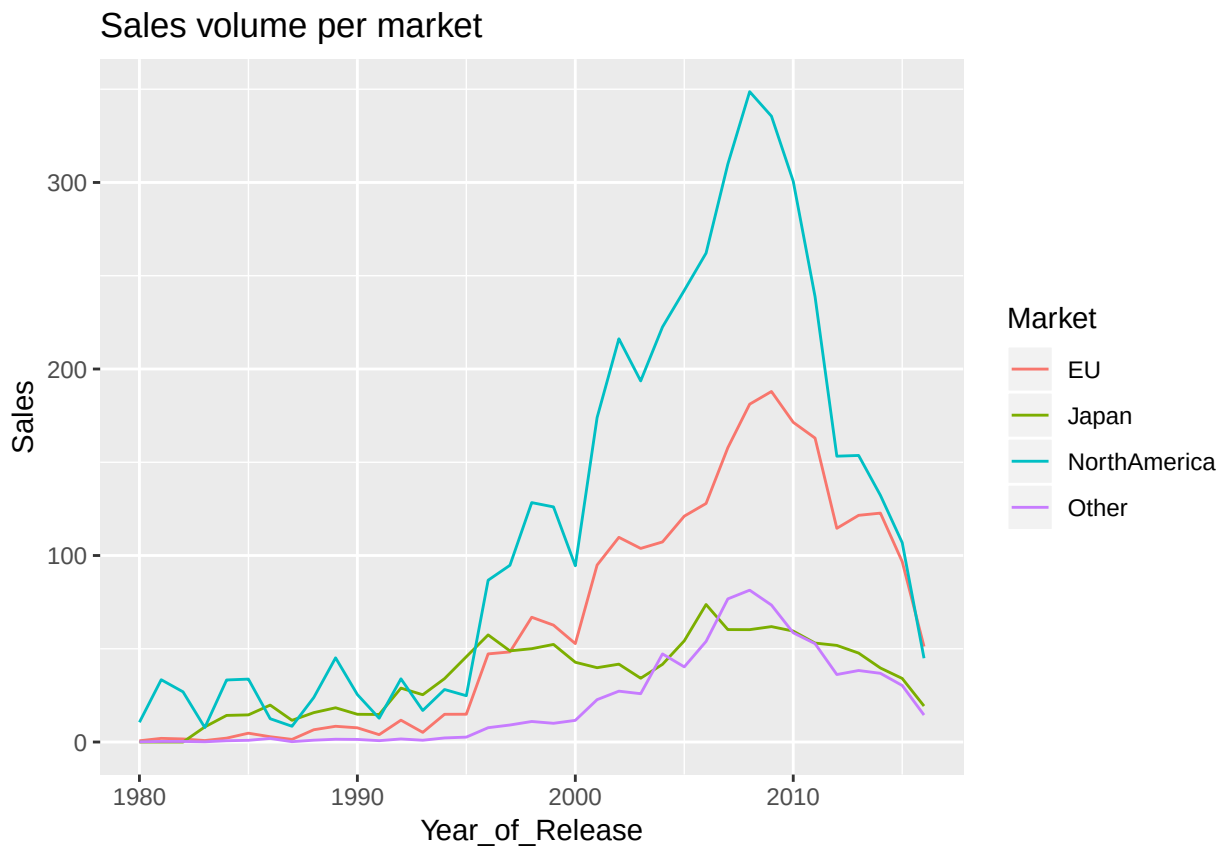

Figure 2: Games sales per region

Let's also display distribution of User Score:

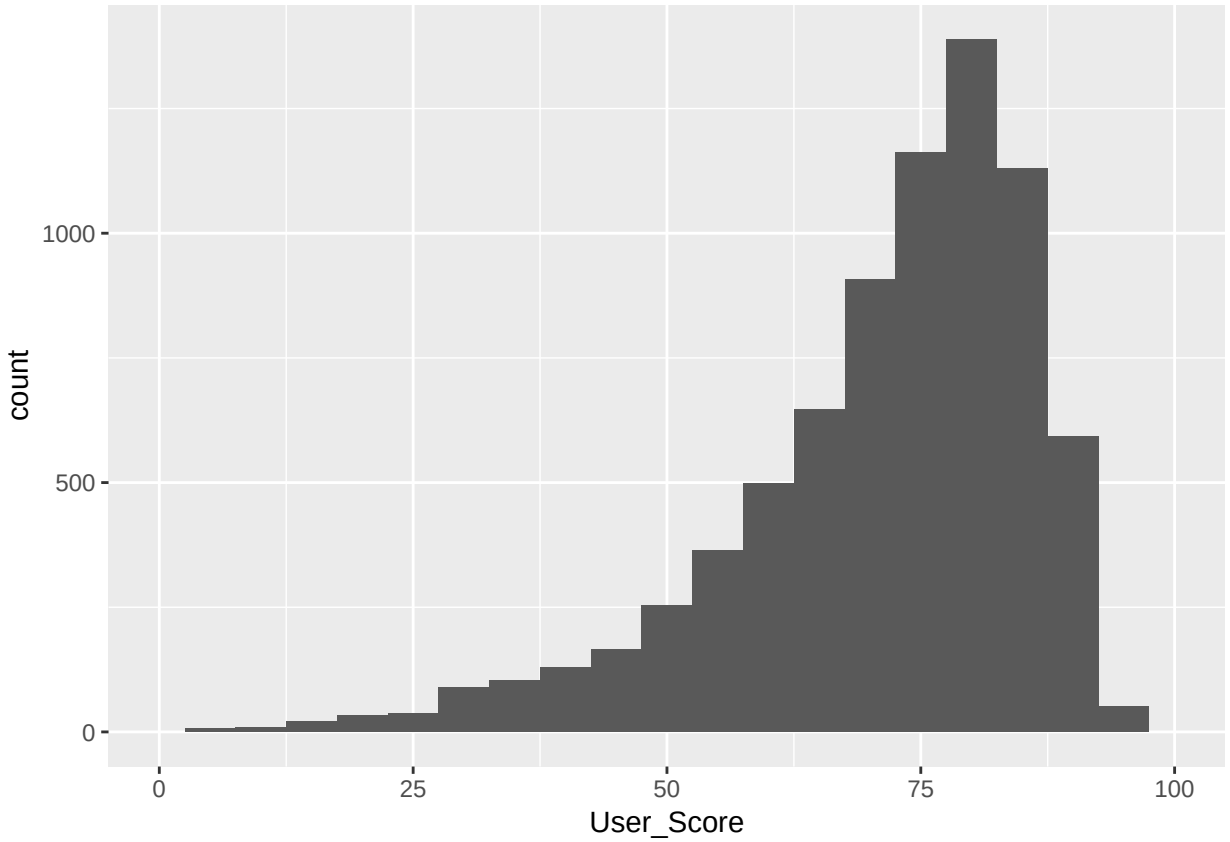

Figure 3: Games sales per region

and Critics score:

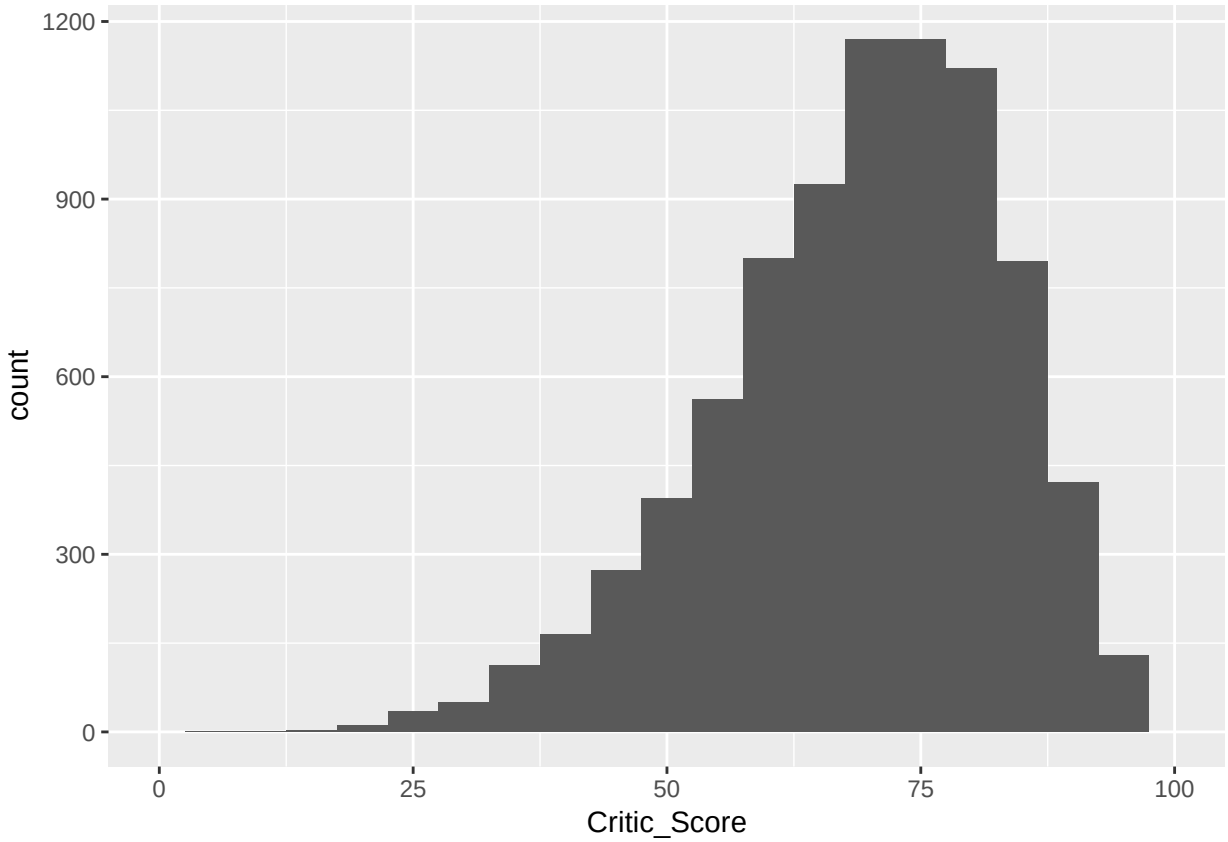

Figure 4: Games sales per region

Let's display top 10 games with highest sales:

```
## Selecting by sumGlobalSales
```

Name	sumGlobalSales
Wii Sports	82.53
Grand Theft Auto V	56.57
Super Mario Bros.	45.31
Tetris	35.84
Mario Kart Wii	35.52
Wii Sports Resort	32.77
Pokemon Red/Pokemon Blue	31.37
Call of Duty: Black Ops	30.82
Call of Duty: Modern Warfare 3	30.59
New Super Mario Bros.	29.80

Top 10 best rated games by users

```
## Selecting by meanUserScore
```

Name	meanUserScore
Boktai: The Sun is in Your Hand	96
Harvest Moon: Friends of Mineral Town	96
Cory in the House	95
Golden Sun: The Lost Age	95
Karnaaj Rally	95
Super Puzzle Fighter II	95
Wade Hixton's Counter Punch	95
Advance Wars 2: Black Hole Rising	94
Castlevania: Symphony of the Night	94
Monster Rancher Advance 2	94
Paper Mario: The Thousand-Year Door	94
Rock 'N Roll Racing	94
Shenmue	94
Skies of Arcadia	94

Top 10 best rates games by critics

```
## Selecting by meanUserScore
```

Name	meanUserScore
Boktai: The Sun is in Your Hand	96
Harvest Moon: Friends of Mineral Town	96
Cory in the House	95
Golden Sun: The Lost Age	95
Karnaaj Rally	95
Super Puzzle Fighter II	95
Wade Hixton's Counter Punch	95
Advance Wars 2: Black Hole Rising	94
Castlevania: Symphony of the Night	94
Monster Rancher Advance 2	94
Paper Mario: The Thousand-Year Door	94
Rock 'N Roll Racing	94
Shenmue	94
Skies of Arcadia	94

Let also also see if there is correlation between User Score and Critic Score:

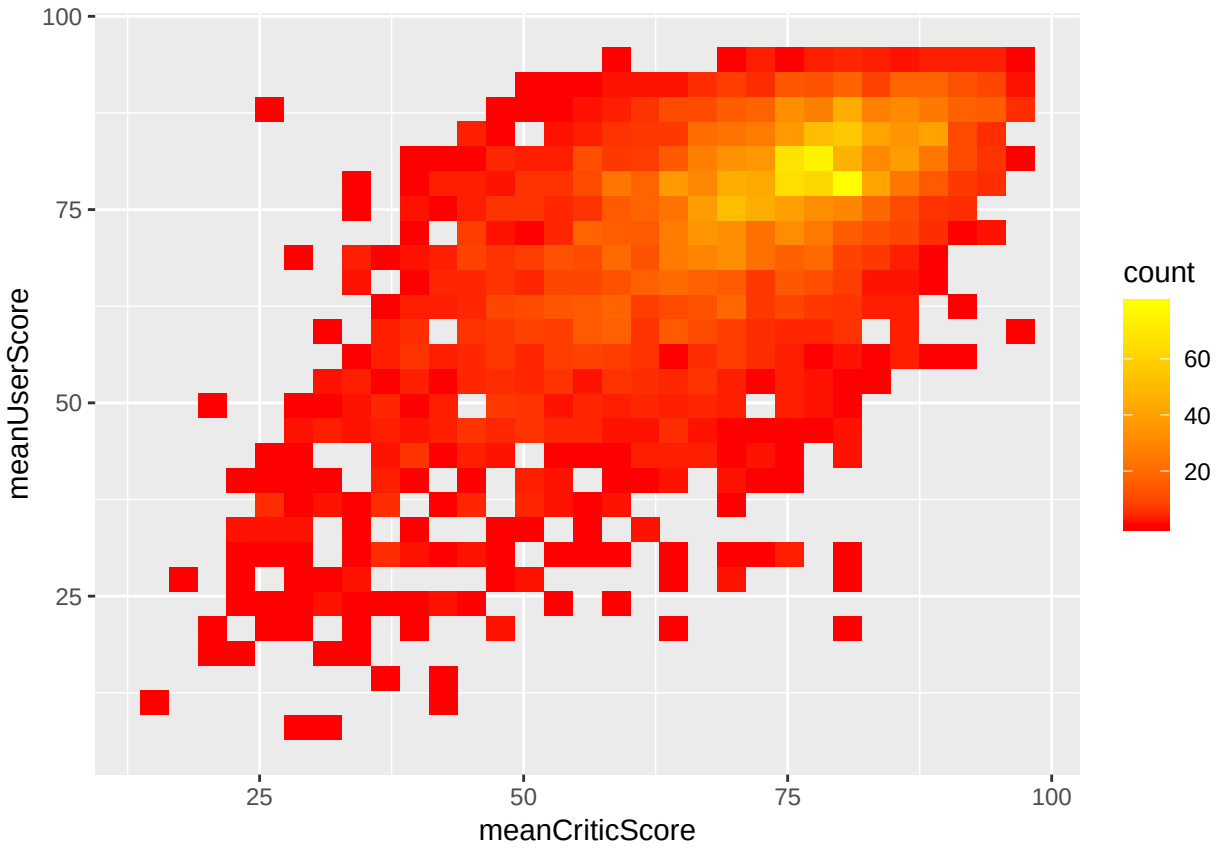

Figure 5: Correlation between Users and Critics score

As we User Score and Critic Score have positive correlation and we can calculate it's value to be: 0.5808778

Looking for features that may have correlation with Global Sales

Let's start with assessing Sales per released game in time:

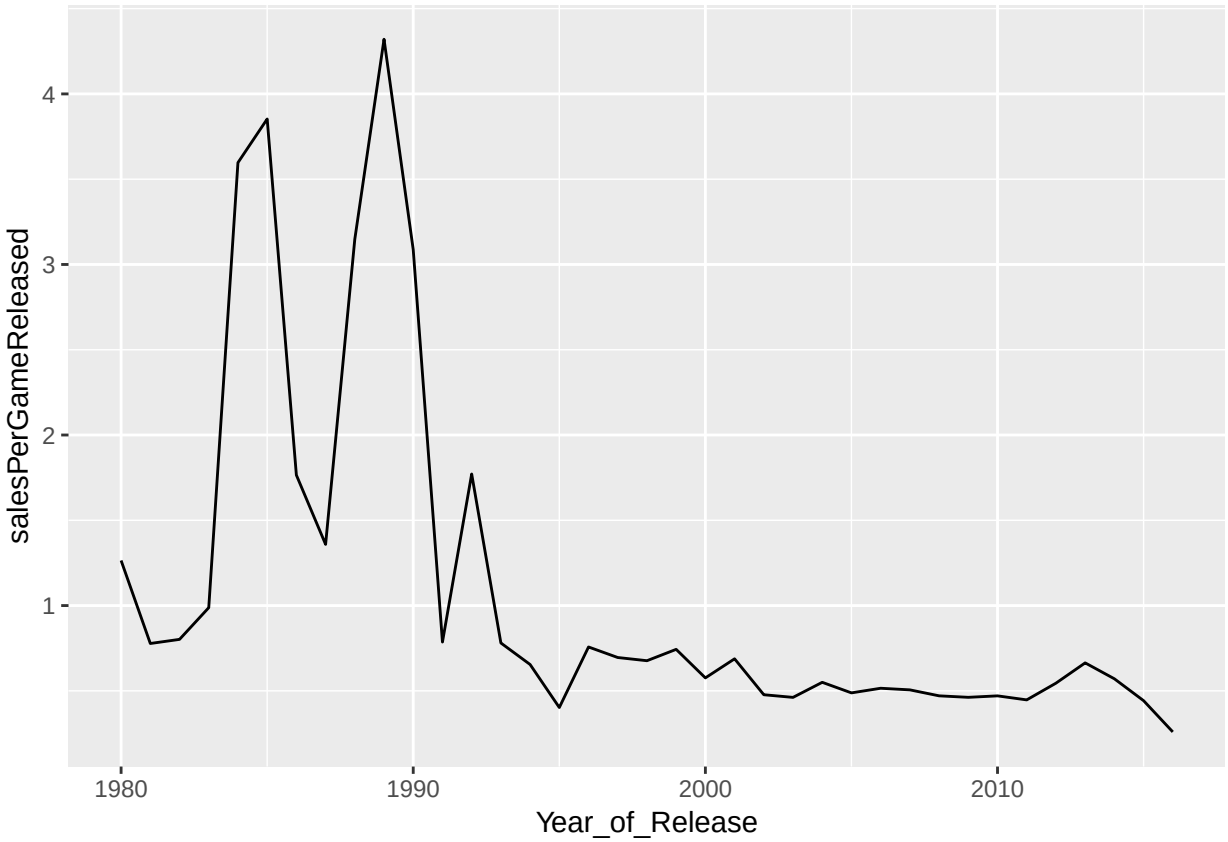

Figure 6: Sales per game released in time

We can see that correlation is quite weak and hence Year_of_Release doesn't seem like a best predictor of sales.

Next let's have a look into Sales per released game depending on Genre:

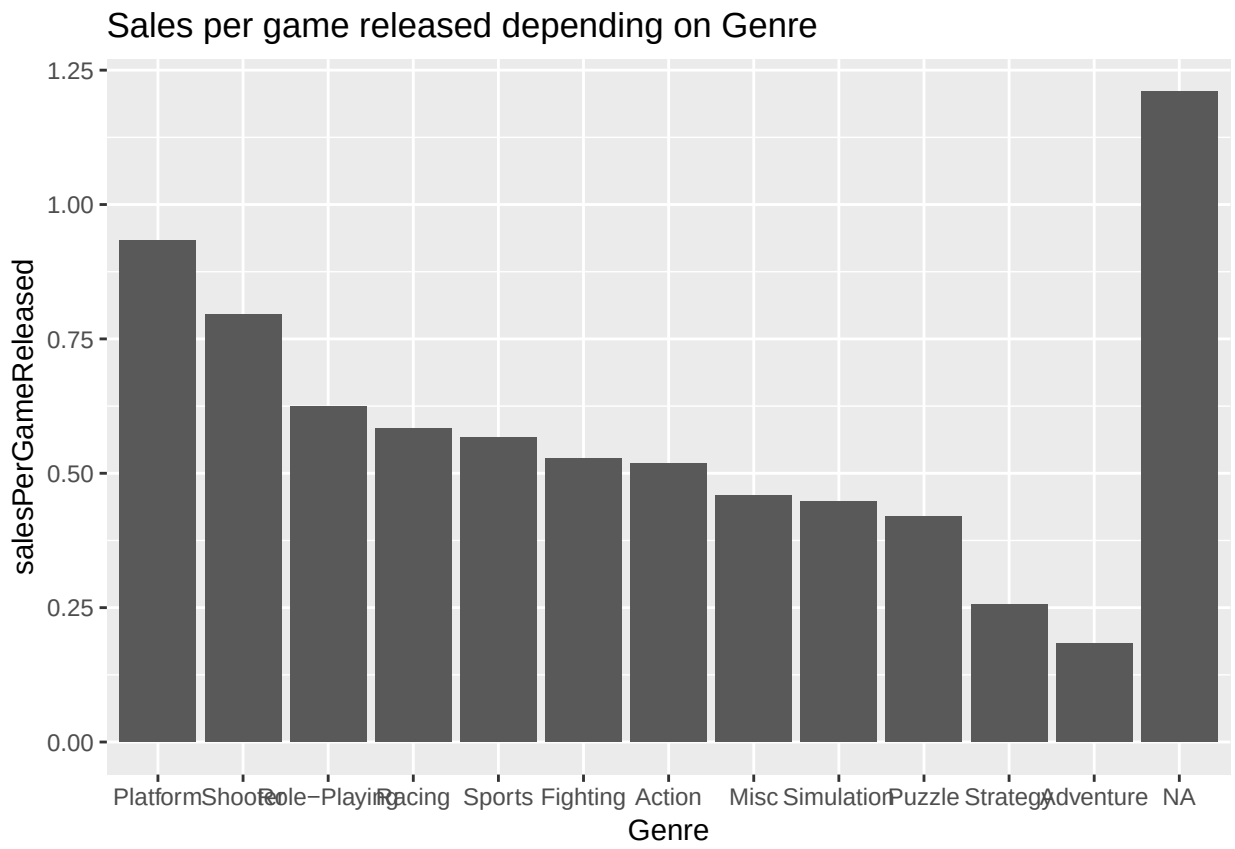

Figure 7: Sales per game released depending on Genre

Diagram indicates that there is some correlation between game Genre and Sales. This is good candidate to become feature in our analysis.

Now let's have a look into Sales per released game depending on Platform:

```
## Selecting by salesPerPlatform
```

Platform	globalSales	count	salesPerPlatform
GB	255.45	98	2.6066327
NES	251.07	98	2.5619388
GEN	30.78	29	1.0613793
SNES	200.05	239	0.8370293
PS4	314.19	392	0.8015051
X360	971.63	1262	0.7699128
2600	97.08	133	0.7299248
PS3	939.43	1331	0.7058077
Wii	908.13	1320	0.6879773
N64	218.88	319	0.6861442

There is some weak correlation visisble as well.

We can also expect that there is extremely strong correlation between Regional Sales and Global Sales. Let's check this by comparing Global Sales and Nort America Sales:

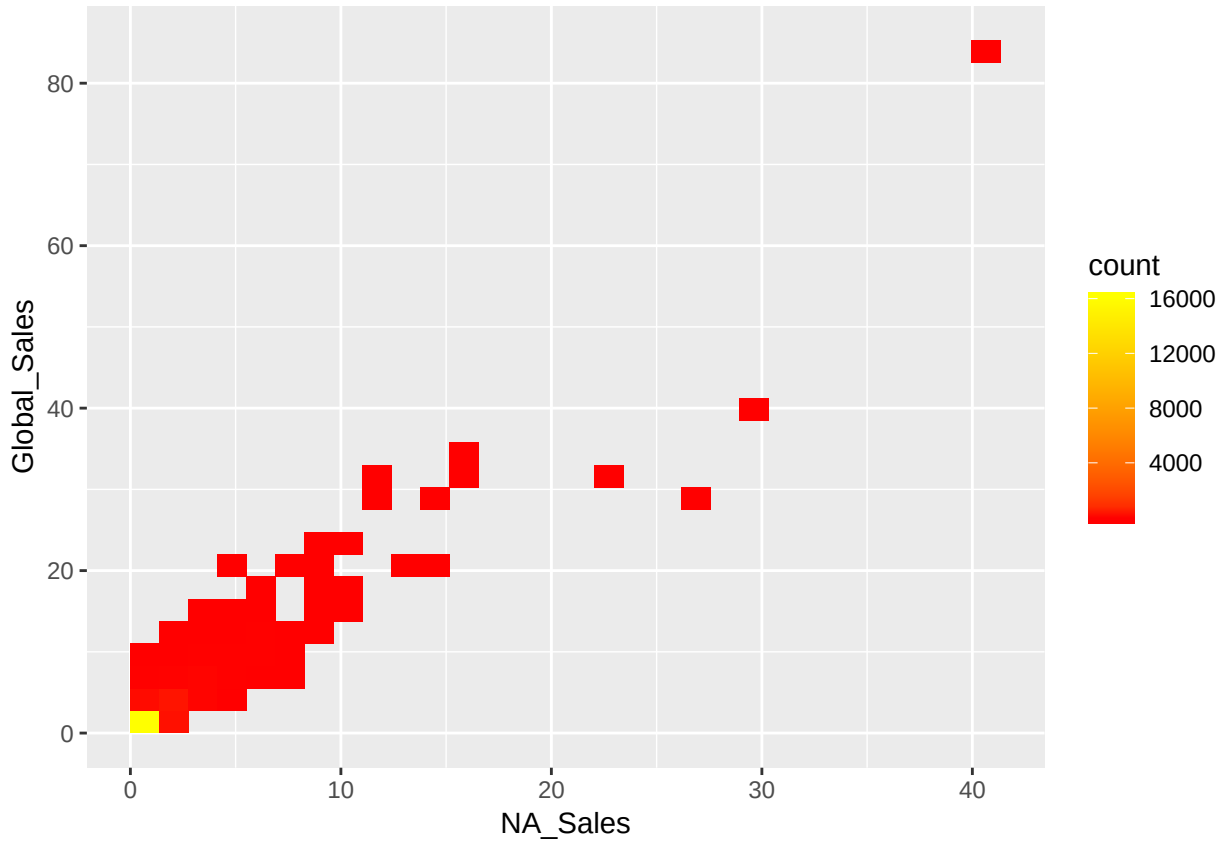

Figure 8: Correlation between North America and Global Sales

As expected, correlation extremely strong. It's value is: 0.9410101.

This is not surprsing, as Regional Sales directly impacts and drives Global Sales. However it's not cause for high Global Sales, they are just correlated.

We can also expect strong correlation between User Count and Global Sales. The better game sales, the more people will play it, the more will rate it. Let's plot it:

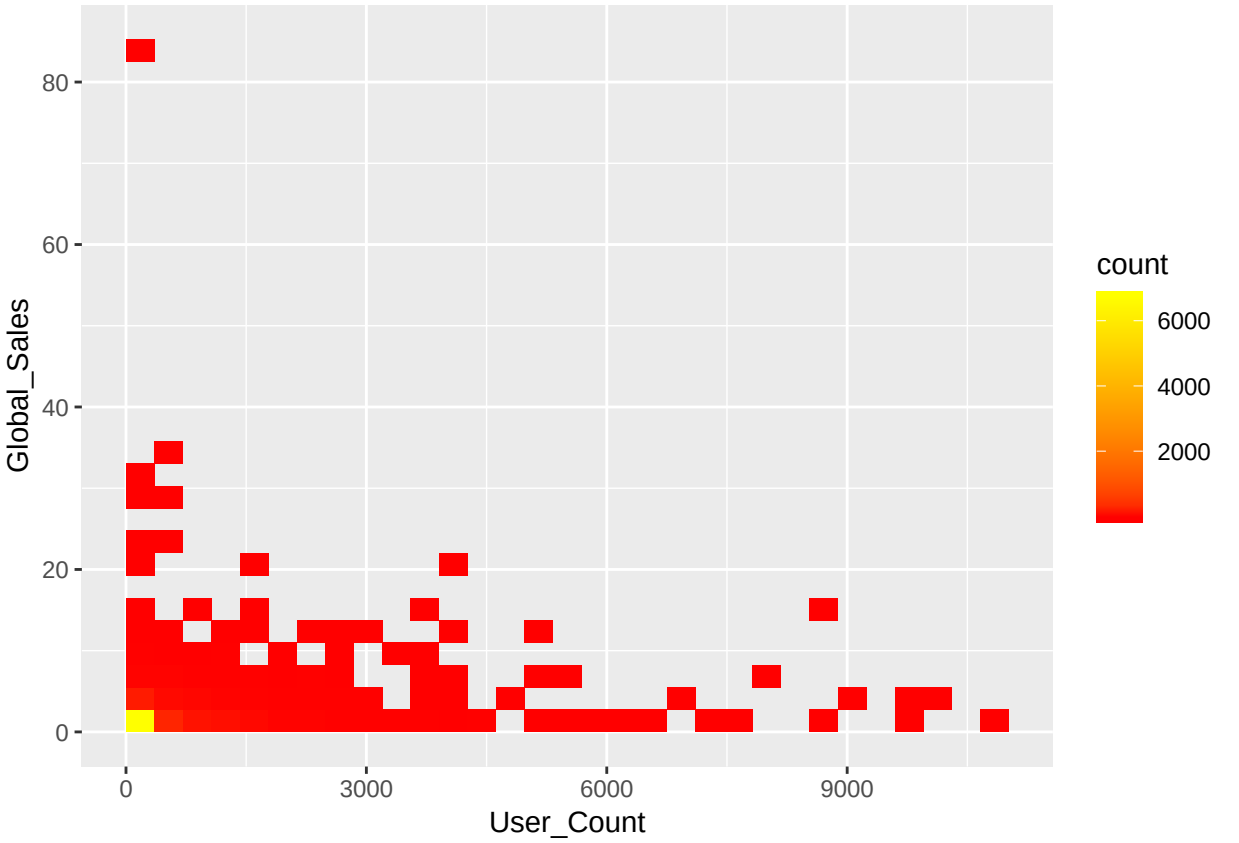

Figure 9: Correlation between Users and Critics score

Surprisingly, correlation is not that strong and equals to only 0.2650115.

Anyhow, high User_Count is not cause of high Global Sales, probably rather it's result. Hence we shouldn't take it as feature for our analysis.

Ok, so we now that we should exclude from our analysis Regional Sales and User Count. We will also exclude observations missing values for our model constructions as this would affect our predictions. Let's do that now:

```
games <- na.omit(games)
games <- subset(games, select = -c(NA_Sales, EU_Sales, JP_Sales, Other_Sales, User_Count))
```

We will select to our analysis following features:

- Platform
- User_Score
- Critic_Score
- Critic_Count
- Genre
- Year_of_Release
- Rating

Building models

First we will set seed to make sure that our analysis results are easily replicable:

```
set.seed(1, sample.kind="Rounding")
```

In order to start model preparation we need first to define function, which will be used to select best performing model. For this we will define MRSE:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

We will also split our dataset into training and validation datasets:

```
validation_index <- createDataPartition(y = games$Global_Sales, times = 1, p = 0.2, list = FALSE)
train <- games[-validation_index,]
validation <- games[validation_index,]
```

Model 0 - Simple mean

Now we are ready to start working on actual models.

As basis for our further analysis we will define simplest possible model which predicts as Global Sale average from all population.

```
mu <- mean(train$Global_Sales)
```

Let's now calculate this model RMSE on training data:

```
model_0_rmse <- RMSE(train$Global_Sales, mu)
model_0_rmse
```

```
## [1] 1.975352
```

Model 1 - Linear model

For our first model we will use linear model:

```
train_lm <- train(Global_Sales ~ Platform + User_Score + Critic_Score + Critic_Count + Genre + Year_of_
                        method = "lm", data = train, na.action = na.exclude)
```

Let's now calculate this model RMSE on training data:

```
getTrainPerf(train_lm)$TrainRMSE
```

```
## [1] 1.818519
```

It's also worth to have a look into variables with highest importance for the model:

```
varImp(train_lm)
```

```
## lm variable importance
##
##   only 20 most important variables shown (out of 37)
##
##                     Overall
## Critic_Count        100.000
## Critic_Score         67.662
## User_Score           30.427
## PlatformPC           23.237
## PlatformXB           20.520
## PlatformWii          20.036
## Year_of_Release      18.447
## GenrePuzzle          15.619
## GenreStrategy        13.361
## PlatformGC           13.012
## PlatformX360          9.391
## GenreMisc             8.676
## PlatformPS            8.636
## PlatformPSV           8.486
## GenreAdventure        8.438
## PlatformPSP           8.349
## PlatformDC            7.776
## PlatformGBA           7.748
## `GenreRole-Playing`   7.030
## PlatformWiiU          5.943
```

Model 2 - Generalized linear model

As our second model we will use GLM:

```
train_glm <- train(Global_Sales ~ Platform + User_Score + Critic_Score + Critic_Count + Genre + Year_of_
                   method = "glm", data = train, na.action = na.exclude)
```

Let's now calculate this model RMSE on training data:

```
getTrainPerf(train_glm)$TrainRMSE
```

```
## [1] 1.663889
```

And here are variables with highest importance for the model:

```
varImp(train_glm)
```

```
## glm variable importance
##
##   only 20 most important variables shown (out of 37)
##
##                     Overall
```

```
## Critic_Count        100.000
## Critic_Score         67.662
## User_Score           30.427
## PlatformPC           23.237
## PlatformXB           20.520
## PlatformWii          20.036
## Year_of_Release      18.447
## GenrePuzzle          15.619
## GenreStrategy        13.361
## PlatformGC           13.012
## PlatformX360          9.391
## GenreMisc             8.676
## PlatformPS            8.636
## PlatformPSV           8.486
## GenreAdventure        8.438
## PlatformPSP           8.349
## PlatformDC            7.776
## PlatformGBA           7.748
## `GenreRole-Playing`   7.030
## PlatformWiiU          5.943
```

Model 3 - Support Vector Machines with Linear Kernel model

As our third model we will use svmLinear:

```
train_svm <- train(Global_Sales ~ Platform + User_Score + Critic_Score + Critic_Count + Genre + Year_of_
                   method = "svmLinear", data = train, na.action = na.exclude)
```

Let's now calculate this model RMSE on training data:

```
getTrainPerf(train_svm)$TrainRMSE
```

```
## [1] 1.784808
```

And here are variables with highest importance for the model:

```
varImp(train_svm)
```

```
## loess r-squared variable importance
##
##                 Overall
## Critic_Count    100.000
## Critic_Score     62.727
## User_Score        9.570
## Genre             4.442
## Rating            2.747
## Platform          2.241
## Year_of_Release   0.000
```

As we can see Year_of_Release feature is not being used for prediction at all.

Model 4 - K-Nearest Neighbors model

For fourth model we will use knn:

```
train_knn <- train(Global_Sales ~ Platform + User_Score + Critic_Score + Critic_Count + Genre + Year_of_
                   method = "knn", data = train, na.action = na.exclude)
```

Let's now calculate this model RMSE on training data:

```
getTrainPerf(train_knn)$TrainRMSE
```

```
## [1] 1.944713
```

And here are variables with highest importance for the model and tested k-values:

```
varImp(train_knn)
```

```
## loess r-squared variable importance
##
##                 Overall
## Critic_Count    100.000
## Critic_Score     62.727
## User_Score        9.570
## Genre             4.442
## Rating            2.747
## Platform          2.241
## Year_of_Release   0.000
```

```
train_knn$results
```

```
##   k     RMSE   Rsquared       MAE    RMSESD RsquaredSD      MAESD
## 1 5 2.073913 0.06484057 0.7225163 0.3659147 0.02894239 0.03256299
## 2 7 1.992476 0.07812731 0.7009785 0.3733560 0.03296553 0.02781611
## 3 9 1.944713 0.08937538 0.6893927 0.3860520 0.03782562 0.02565078
```

Model 5 - Random Forest

As our fifth model we wll use Random Forest:

```
train_rf <- train(Global_Sales ~ Platform + User_Score + Critic_Score + Critic_Count + Genre + Year_of_
                  method = "rf", data = train, na.action = na.exclude, ntree=50, metric="RMSE", trContr
```

Let's now calculate this model RMSE on training data and let's review achieved results depending on mtry parameter:

```
getTrainPerf(train_rf)$TrainRMSE
```

```
## [1] 1.558968
```

```
train_rf$results
```

```
##   mtry     RMSE  Rsquared       MAE    RMSESD RsquaredSD      MAESD
## 1    2 1.720868 0.2495862 0.7178909 0.8278953 0.09639389 0.07587058
## 2   27 1.558968 0.3158782 0.5893770 0.8360730 0.14548938 0.08024635
## 3   52 1.613416 0.2934143 0.6062689 0.8216650 0.14653557 0.08185428
```

Model 6 - Random Forest After Tuning model

As our final model we will try to optimize Random Forest.

First we will define tuning grid and variable to store results of optimization:

```
mtry = c(5:15)
rf_grid <- data.frame(mtry)
rf_grid
```

Now we will run our model multiple times to find optimal parameters: mtry & ntree:

```
modellist <- list()
for (numtree in seq(5,100,by=5)) {
  train_trf <- train(Global_Sales ~ Platform + User_Score + Critic_Score + Critic_Count + Genre + Year_
                     method = "rf", data = train, na.action = na.exclude, ntree=numtree, tuneGrid=rf_gr
  tuning_results[nrow(tuning_results) + 1,] = c(numtree, train_trf$results$RMSE)
  key <- toString(numtree)
  modellist[[key]] <- train_trf
}
```

Now we can display achieved RMSE depending on mtry & ntree parameters:

```
tuning_results
```

```
##    ntree        5        6        7        8        9       10       11
## 1      5 1.671688 1.668069 1.645945 1.738907 1.710339 1.710606 1.748665
## 2     10 1.620872 1.592712 1.608683 1.596639 1.594493 1.596689 1.649082
## 3     15 1.619133 1.580520 1.577611 1.618228 1.578264 1.579424 1.593656
## 4     20 1.593376 1.573454 1.581296 1.576262 1.569238 1.577094 1.568635
## 5     25 1.577129 1.561741 1.574801 1.568738 1.567336 1.568517 1.569372
## 6     30 1.608619 1.597551 1.584450 1.581529 1.581642 1.567145 1.594646
## 7     35 1.595607 1.584586 1.569649 1.585575 1.568869 1.578723 1.570576
## 8     40 1.574967 1.567150 1.555963 1.561097 1.559151 1.569068 1.565127
## 9     45 1.598163 1.585510 1.575768 1.576147 1.573519 1.572729 1.570803
## 10    50 1.564860 1.554885 1.550593 1.547130 1.549237 1.553641 1.544765
## 11    55 1.559932 1.547086 1.541526 1.544454 1.537890 1.538553 1.549222
## 12    60 1.582490 1.564141 1.553641 1.545907 1.550188 1.557795 1.555927
## 13    65 1.565335 1.552506 1.535081 1.539828 1.528256 1.537899 1.544871
## 14    70 1.599920 1.574666 1.571997 1.568883 1.560612 1.568787 1.561018
## 15    75 1.583773 1.574197 1.561783 1.559294 1.559925 1.552696 1.549327
## 16    80 1.583126 1.567985 1.553866 1.557267 1.555161 1.550290 1.552762
## 17    85 1.604961 1.578488 1.582596 1.569080 1.566481 1.561758 1.570756
## 18    90 1.578660 1.565909 1.546230 1.550892 1.554669 1.546020 1.547124
```

```
## 19    95 1.560978 1.549366 1.539027 1.527371 1.525652 1.526881 1.531004
## 20   100 1.564483 1.544298 1.548739 1.533179 1.547018 1.535141 1.539781
##          12       13       14       15
## 1  1.689621 1.668051 1.715625 1.721861
## 2  1.623615 1.650364 1.586959 1.616575
## 3  1.611789 1.604804 1.608857 1.597067
## 4  1.571986 1.588354 1.562701 1.593497
## 5  1.609054 1.576486 1.577314 1.575129
## 6  1.596852 1.597424 1.611928 1.595501
## 7  1.588700 1.566062 1.572196 1.591365
## 8  1.569177 1.565058 1.581701 1.556760
## 9  1.593496 1.582524 1.589151 1.593121
## 10 1.552367 1.565795 1.563914 1.569999
## 11 1.545610 1.558814 1.544329 1.550923
## 12 1.562047 1.561912 1.569365 1.566061
## 13 1.532787 1.554144 1.543090 1.555111
## 14 1.564746 1.572329 1.560541 1.572384
## 15 1.560858 1.558778 1.555809 1.568041
## 16 1.559829 1.561705 1.560564 1.559876
## 17 1.573808 1.584519 1.576816 1.570374
## 18 1.558678 1.552815 1.563720 1.564357
## 19 1.533780 1.540069 1.534047 1.547533
## 20 1.541787 1.556829 1.547142 1.542862
```

Now we can plot graph presenting achieved RMSE depending on Random Forest tuning parameters:

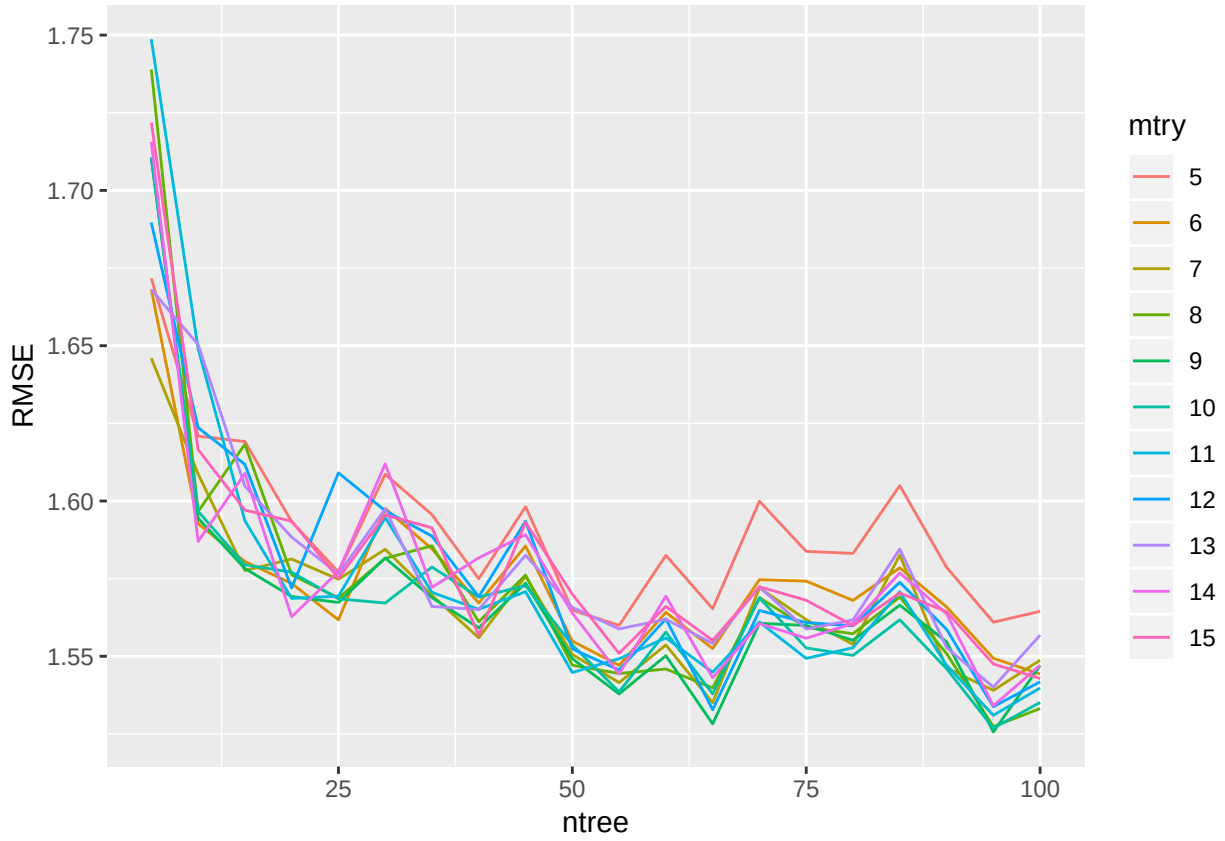

Figure 10: mtry & ntree parameters impact on achieved MRSE

Now we can display optimal RMSE and parameters for which it was achieved:

```
min(tr$RMSE)
```

```
## [1] 1.525652
```

```
tr[which.min(tr$RMSE),]
```

```
##    ntree mtry     RMSE
## 99    95    9 1.525652
```

Results

On our training dataset we have achieved following results:

method	RMSE
Model 0 - Simple mean	1.975352
Model 1 - Linear model	1.818519
Model 2 - Generalized Linear model	1.663889

method	RMSE
Model 3 - Support Vector Machines with Linear Kernel model	1.784808
Model 4 - k-Nearest Neighbors model	1.944713
Model 5 - Random Forest	1.558968
Model 6 - Random Forest After Tuning model	1.525652

As our best performing model we select last Model 6 - Tuned Random Forest model.

Now when we have selected model together with parameters it's time to test it performance on our validation dataset.

For this let's code our final model:

```
optimal_mtry <- as.numeric(as.character(tr[which.min(tr$RMSE),2]))
optimal_mtry
```

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

```
optimal_ntree <- as.numeric(as.character(tr[which.min(tr$RMSE),1]))
optimal_ntree
```

```
## [1] 95
```

```
optimal_rf_grid <- data.frame(mtry = optimal_mtry)
train_final <- train(Global_Sales ~ Platform + User_Score + Critic_Score + Critic_Count + Genre + Year_
                     method = "rf", data = train, na.action = na.exclude, ntree = optimal_ntree, tuneGrid
```

and let's predict Global Sales based on our validation data:

```
predicted <- predict(train_final, validation)
```

Finally we can calculate RMSE on our validation data:

```
RMSE(validation$Global_Sales, predicted)
```

```
## [1] 1.45493
```

Conclusion

Video Games Sales dataset has been explored and analysed.

Regional sales data and user count have been exluded as features, as they are not causation of Global Sales.

In order to best predict Global Sales based on available features first Reference Model (Model 0) has been created.

Then 6 models have been implemented and their results analysed.

The best performance offers last model - Random Forest with optimized parameters.

It allowed us to achieve RMSE on validation dataset of 1.4549304.

Reference / simplest model on validation data allows to achieve: 1.913926

In comparision to the reference / simplest model this is improvement of: 0.24%.

This result is far from being sufficient for real live sales prediction. This is not surprising as there are many factors which impact games sales and in our model we were able to include only 7 features.

In order to improve our predictions we should have more information, like for example: development budget, marketing spendings etc.

Our model could be improved as well, for example by applying Matrix Factorization. We should cluster similar titles (like for example games from same series) and apply this effects to our predictions.