Video Game Sales with Ratings project

Contents

1 Introduction

1.1 Project scope

In this project I explore and analyze *Video Game Sales with Ratings* data set made by Rush Kirubi. For the analysis sake the data set is stored on my GitHub repository (as well as this report, full code in .R and .rmd file), but original can be found on Kaggle.

Besides data analysis I will try to find best possible prediction model for Global and North American Sales.

Libraries used for the sake of this project are: *tidyverse*, *funModeling*, *scales*, *caret*, *corrgram*, *ggpubr*, *wordcloud*, *tm* and *gridExtra*. For Cubist and NNLS (Non-Negative Least Squares) models additional *cubist* and *nnls* packages needs to be installed.

In this project I've tried to create all plots in pallets that can be perceived by people affected by color vision deficiency, following *RColorBrewer* palette guidelines.

This pdf report contains most code for any data exploration and machine learning I will be doing, but some other transformations and code for plots creations have been omitted for report clarity. You can find full code in the *Appendix/Full code* session, or .rmd and .R files on my GitHub repository mentioned above. Small portions of the code presented in .rmd table might be different from .R file, for the sake of presenting plots and tables in final pdf report.

1.2 Data overview

Video Games Sales with Ratings data set was imported to *video_sales* object, upon which I will be conducting the analysis (raw data set is saved in *raw.videosales* object in case reference would be needed).

Taking first look into the data set by using *glimpse* I identify 16 variables with 16,719 records.

```
## Rows: 16,719
## Columns: 16
## $ Name            <chr> "Wii Sports", "Super Mario Bros.", "Mario Kart Wii"...
## $ Platform        <chr> "Wii", "NES", "Wii", "Wii", "GB", "GB", "DS", "Wii"...
## $ Year_of_Release <chr> "2006", "1985", "2008", "2009", "1996", "1989", "20...
## $ Genre           <chr> "Sports", "Platform", "Racing", "Sports", "Role-Pla...
## $ Publisher       <chr> "Nintendo", "Nintendo", "Nintendo", "Nintendo", "Ni...
## $ NA_Sales        <dbl> 41.36, 29.08, 15.68, 15.61, 11.27, 23.20, 11.28, 13...
## $ EU_Sales        <dbl> 28.96, 3.58, 12.76, 10.93, 8.89, 2.26, 9.14, 9.18, ...
## $ JP_Sales        <dbl> 3.77, 6.81, 3.79, 3.28, 10.22, 4.22, 6.50, 2.93, 4....
## $ Other_Sales     <dbl> 8.45, 0.77, 3.29, 2.95, 1.00, 0.58, 2.88, 2.84, 2.2...
## $ Global_Sales    <dbl> 82.53, 40.24, 35.52, 32.77, 31.37, 30.26, 29.80, 28...
## $ Critic_Score    <int> 76, NA, 82, 80, NA, NA, 89, 58, 87, NA, NA, 91, NA,...
## $ Critic_Count    <int> 51, NA, 73, 73, NA, NA, 65, 41, 80, NA, NA, 64, NA,...
## $ User_Score      <chr> "8", NA, "8.3", "8", NA, NA, "8.5", "6.6", "8.4", N...
## $ User_Count      <int> 322, NA, 709, 192, NA, NA, 431, 129, 594, NA, NA, 4...
## $ Developer       <chr> "Nintendo", NA, "Nintendo", "Nintendo", NA, NA, "Ni...
## $ Rating          <chr> "E", NA, "E", "E", NA, NA, "E", "E", "E", NA, NA, "...
```

According to Kirubi (creator of the data set) and the glimpse above, the 16 variables corresponds to:

- *Name* - name of the game
- *Platform* - platform on which the game was released
- *Year_of_Release* - year of release of the game

- *Genre* - game's genre (category)
- *Publisher* - publisher of the game
- *NA_Sales* - millions of units of selected game sold in North America, as per Dec 2016 data
- *EU_Sales* - millions of units of selected game sold in European Union, as per Dec 2016 data
- *JP_Sales* - millions of units of selected game sold in Japan, as per Dec 2016 data
- *Other_Sales* - millions of units of selected game sold in the rest of the world, as per Dec 2016 data
- *Global_Sales* - millions of units of selected game sold in the whole world, as per Dec 2016 data
- *Critics_Score* - aggregated score by critics, compiled by *Metacritic* site
- *Critic_Count* - number of critics participating in scoring
- *User_Score* - aggregated score of users compiled by *Metacritic* site
- *User_Count* - number of users participating in scoring
- *Developer* - official developer responsible for game creation
- *Rating* - official North America rating assigned by *ESRB* (Entertainment Software Rating Board)

First, I check if any of the game titles is recorded more than once:

```
duplicated(video_sales) %>% sum()
```

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

There are no duplicates in this data set, so I can assume that all 16,719 rows are unique games. Let's see if we can identify any significant issues and outliers, by using a *summary* function.

```
##      Name              Platform         Year_of_Release       Genre
##  Length:16719       Length:16719       Length:16719       Length:16719
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Publisher            NA_Sales          EU_Sales         JP_Sales
##  Length:16719       Min.   : 0.0000   Min.   : 0.000   Min.   : 0.0000
##  Class :character   1st Qu.: 0.0000   1st Qu.: 0.000   1st Qu.: 0.0000
##  Mode  :character   Median : 0.0800   Median : 0.020   Median : 0.0000
##                     Mean   : 0.2633   Mean   : 0.145   Mean   : 0.0776
##                     3rd Qu.: 0.2400   3rd Qu.: 0.110   3rd Qu.: 0.0400
##                     Max.   :41.3600   Max.   :28.960   Max.   :10.2200
##
##   Other_Sales        Global_Sales      Critic_Score    Critic_Count
##  Min.   : 0.00000   Min.   : 0.0100   Min.   :13.00   Min.   :  3.00
##  1st Qu.: 0.00000   1st Qu.: 0.0600   1st Qu.:60.00   1st Qu.: 12.00
##  Median : 0.01000   Median : 0.1700   Median :71.00   Median : 21.00
##  Mean   : 0.04733   Mean   : 0.5335   Mean   :68.97   Mean   : 26.36
##  3rd Qu.: 0.03000   3rd Qu.: 0.4700   3rd Qu.:79.00   3rd Qu.: 36.00
##  Max.   :10.57000   Max.   :82.5300   Max.   :98.00   Max.   :113.00
##                                       NA's   :8582    NA's   :8582
##   User_Score          User_Count       Developer            Rating
##  Length:16719       Min.   :    4.0   Length:16719       Length:16719
##  Class :character   1st Qu.:   10.0   Class :character   Class :character
##  Mode  :character   Median :   24.0   Mode  :character   Mode  :character
##                     Mean   :  162.2
##                     3rd Qu.:   81.0
```

```
##                   Max.   :10665.0
##                   NA's   :9129
```

None of the character variables, like Name, Platform or Publisher, can be analyzed with standard numerical functions, so it might be worth to consider changing them into other format. In all sales besides total (*Global_Sales*) there is a minimum 0, which might be an actual *NA* - also worth considering. Maximum value in *Global_Sales* is significantly higher than maximum values in sales of sub-sectors, so it needs to be investigated if it's not actual error.

More than that, it is already visible from glimpse that *Critic_Score*, *Critic_Count* and *User_Count* have many *NA* values. It is highly possible, that had *User_Score* been also an integer, it would be also filled with *NA*s - another point to investigate further in the data cleaning and exploration. Let's take a closer look into ratio of *NA*s and unique values number, by using a *status* function.

Table 1: Video Sales - zeros, NAs and unique values of variables

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
Name	0	0.0000000	2	0.0001196	0	0	character	11562
Platform	0	0.0000000	0	0.0000000	0	0	character	31
Year_of_Release	0	0.0000000	0	0.0000000	0	0	character	40
Genre	0	0.0000000	2	0.0001196	0	0	character	12
Publisher	0	0.0000000	0	0.0000000	0	0	character	582
NA_Sales	4511	0.2698128	0	0.0000000	0	0	numeric	402
EU_Sales	5874	0.3513368	0	0.0000000	0	0	numeric	307
JP_Sales	10515	0.6289252	0	0.0000000	0	0	numeric	244
Other_Sales	6604	0.3949997	0	0.0000000	0	0	numeric	155
Global_Sales	0	0.0000000	0	0.0000000	0	0	numeric	629
Critic_Score	0	0.0000000	8582	0.5133082	0	0	integer	82
Critic_Count	0	0.0000000	8582	0.5133082	0	0	integer	106
User_Score	1	0.0000598	6704	0.4009809	0	0	character	96
User_Count	0	0.0000000	9129	0.5460255	0	0	integer	888
Developer	0	0.0000000	6623	0.3961361	0	0	character	1696
Rating	0	0.0000000	6769	0.4048687	0	0	character	8

Seeing the table above, it is clear that it's not only Critic and User variables that have an issue with high *NA*s ratio, but also *Developer* and *Rating* variables. While a lot of prediction models can handle *NA*s in the variables, and exclude them from their algorithms, it can create significant bias. That's why any variable with high (definitely above 50%) ratio of *NA*s need to be treated carefully. Easy way would be to exclude them entirely, but since those ratios are high, we would be effectively removing half of our data set. That's why deeper analysis must be conducted and alternative ways (like median imputing) considered.

Status function is also showing us how many unique values each variable has. While it is tempting to change most of the character variables to factors, if factor has high number of levels it may cause overfitting in later stages of modeling (especially in decision trees with low cardinality). More than that, changing variables with too low unique numbers to factors may cause malfunctioning and errors in modeling due to variables having near to zero variance in training sets. That's why I will refrain from transforming them to factors all at once and analyze each case one by one.

2 Data exploration, visualization and cleaning

2.1 Global and regional sales variables

I will begin the analysis from the main output, which is total and regional sales, counted as millions of units. Below there is a density plot of *Global_Sales* variable. It is immediately obvious that the data are right-skewed, with very long tail.

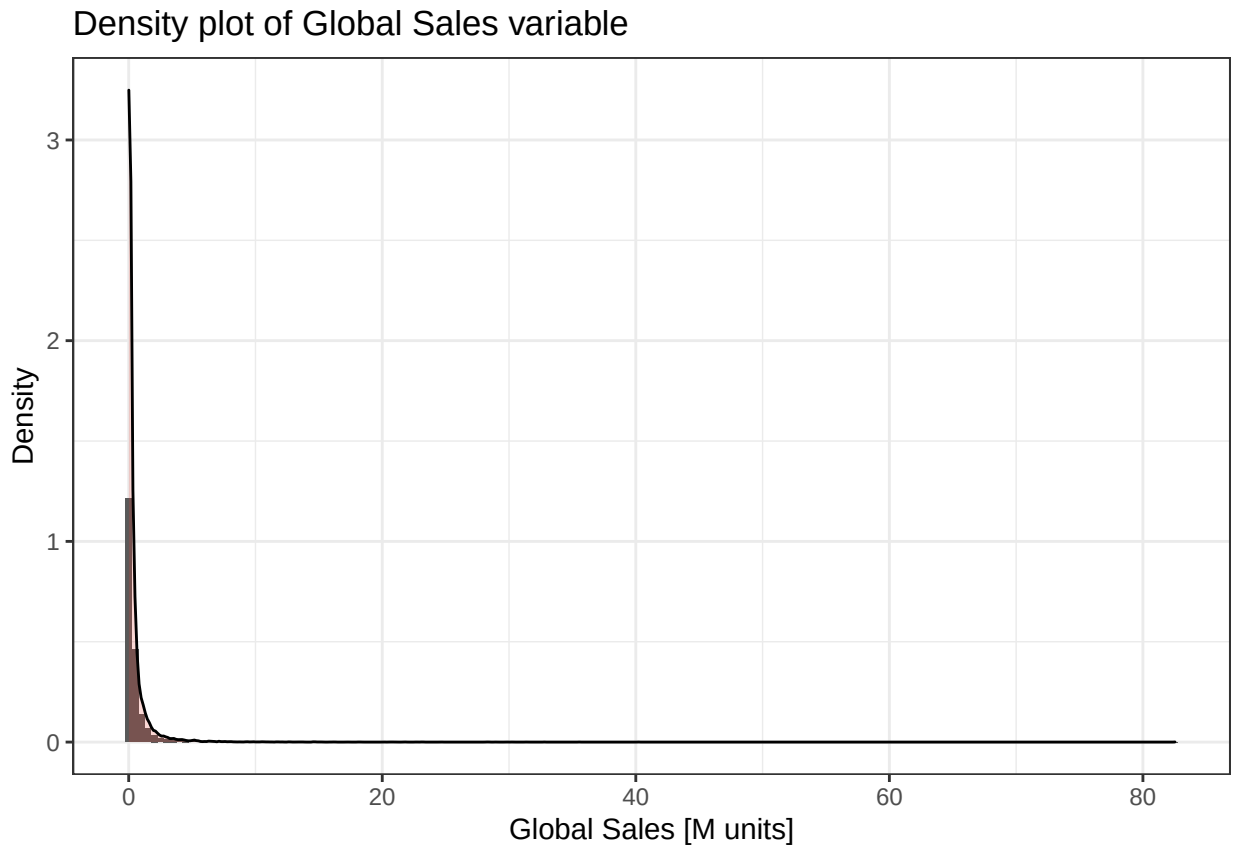

Figure 1: Density of Global Sales

Analyzing top 10 global sales and their corresponding games we see that mainly one game (*Wii Sports*) is responsible for the long tail.

```
video_sales %>%
  select(Name,Year_of_Release,Platform, Genre, User_Count, Global_Sales) %>%
  arrange(desc(Global_Sales)) %>%
  head(10)
```

Table 2: Top 10 best selling games globally

Name	Year_of_Release	Platform	Genre	User_Count	Global_Sales
Wii Sports	2006	Wii	Sports	322	82.53
Super Mario Bros.	1985	NES	Platform	NA	40.24
Mario Kart Wii	2008	Wii	Racing	709	35.52
Wii Sports Resort	2009	Wii	Sports	192	32.77
Pokemon Red/Pokemon Blue	1996	GB	Role-Playing	NA	31.37
Tetris	1989	GB	Puzzle	NA	30.26
New Super Mario Bros.	2006	DS	Platform	431	29.80
Wii Play	2006	Wii	Misc	129	28.92
New Super Mario Bros. Wii	2009	Wii	Platform	594	28.32
Duck Hunt	1984	NES	Shooter	NA	28.31

Wii Sports (the top seller) was released on *Wii* platform and it's a sports game. Seeing as there are other games in top 10 that are sport type and were released on *Wii*, it doesn't seem like the sales output for this title is an error. I decided to leave this title in the data set. Below is the density plot only up to 3rd quartile, so without the right tail created by best selling games. The right-skewedness is now clearly visible, with almost all game titles falling below the overall mean sale being 0.5M units, and highest density around 0.3M global sales. This means that most games sell over x275 less than the best selling game recorded in data set.

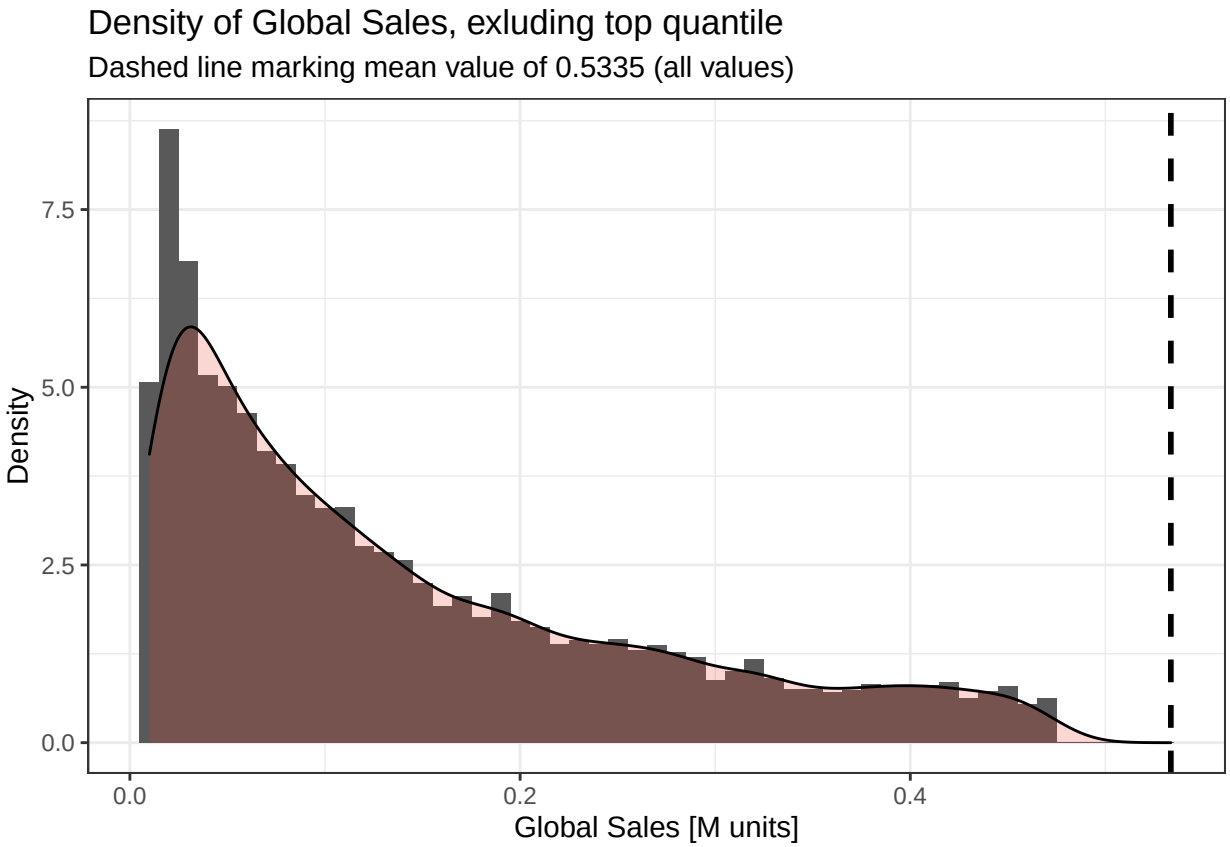

Figure 2: Density of Global Sales, exluding top quantile

Next let's analyze how games were faring through years. Below I explore Global Sales aggregated for all titles per each year on one plot, sales per one title each year on the second one, and the general number of titles released every year. In order to do that, I will change the format of the *Year_of_Release* to numeric in the whole data set (as generally years should be consider numeric or date format).

```
video_sales <- video_sales %>%
  mutate(Year_of_Release=as.numeric(Year_of_Release))
```

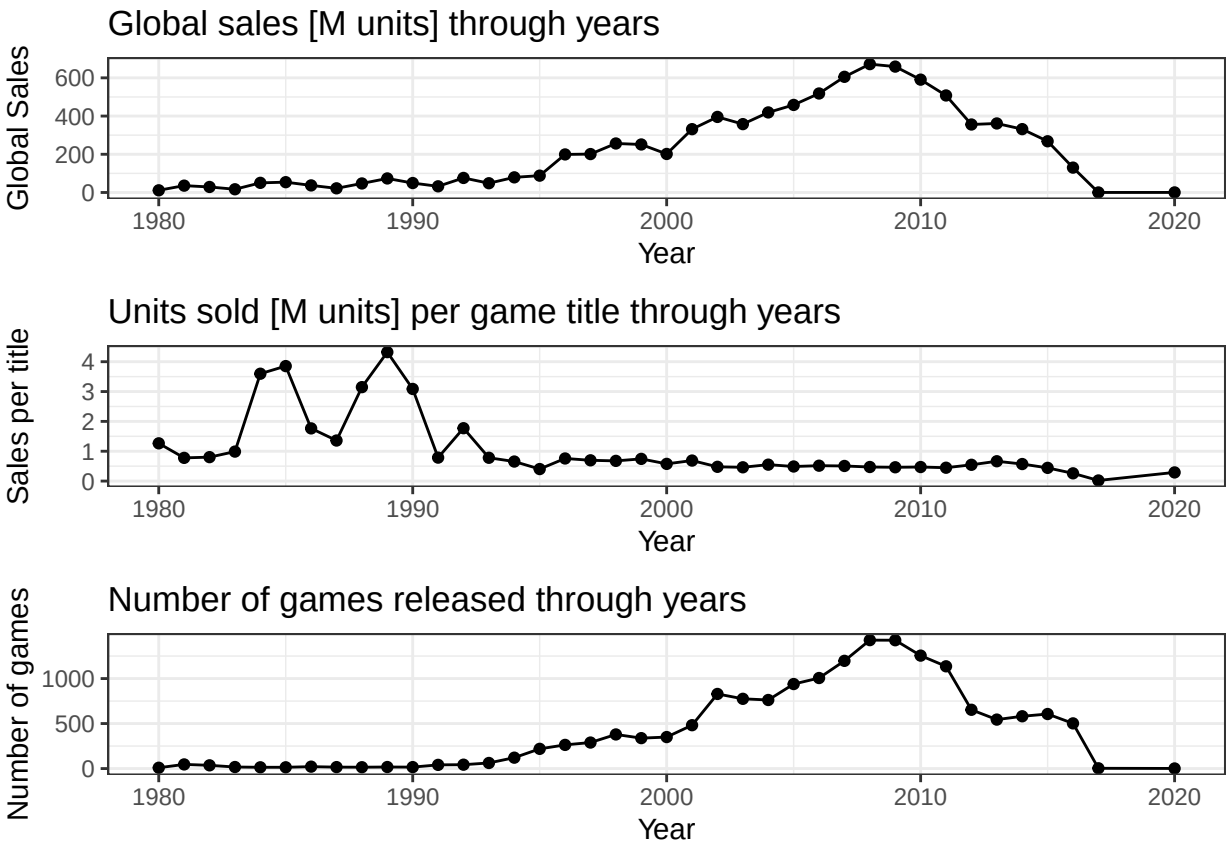

Figure 3: Global Sales, units sold and number of games released through years

There is a visible trend starting sometime in the mid-90s. While the total sales has started to increase, at the same time the number of sold copies per title has been dropping. This leads to conclusion that after mid-90s there has been high increase in number of games on the market (as seen clearly on third plot on Figure 3), where most of them were having much smaller profit than in the past. There is a big peak in Global Sales around 2006 - again, contributed to best selling game *Wii Sport* and similar games on *Wii* platform published during the same period.

Another thing that needs to be re-checked are the values after the year 2016.

Table 3: Number of games released in last 10 recorded years

Year_of_Release	n
2020	1
2017	3
2016	502
2015	606
2014	581
2013	544
2012	653
2011	1136
2010	1255
2009	1426

It seems that not only there is a gap between 2017 and 2020, but the numbers of 2017 and 2020 seems very off compared to previous years. More than that, original data set created by Kirubi has been created as of December 2016, so it is safe to say that all records after that date can be treated as errors and therefore erased.

```
video_sales <- video_sales %>% filter(Year_of_Release < 2017)
```

Lastly let's analyze how the sales are distributed between different regions. On Figure 4 below we can see this distribution through years. At the beginning of 1980 the most dominant market was North American one, but it soon had too compete with Japan. Japanese market tried to gain higher share up until middle of 1990, when they started to significantly lose again with North America region. But all across those years EU and Other markets have been growing step-by-step without much disruptions, and at the present times EU market has almost equal share to the NA market.

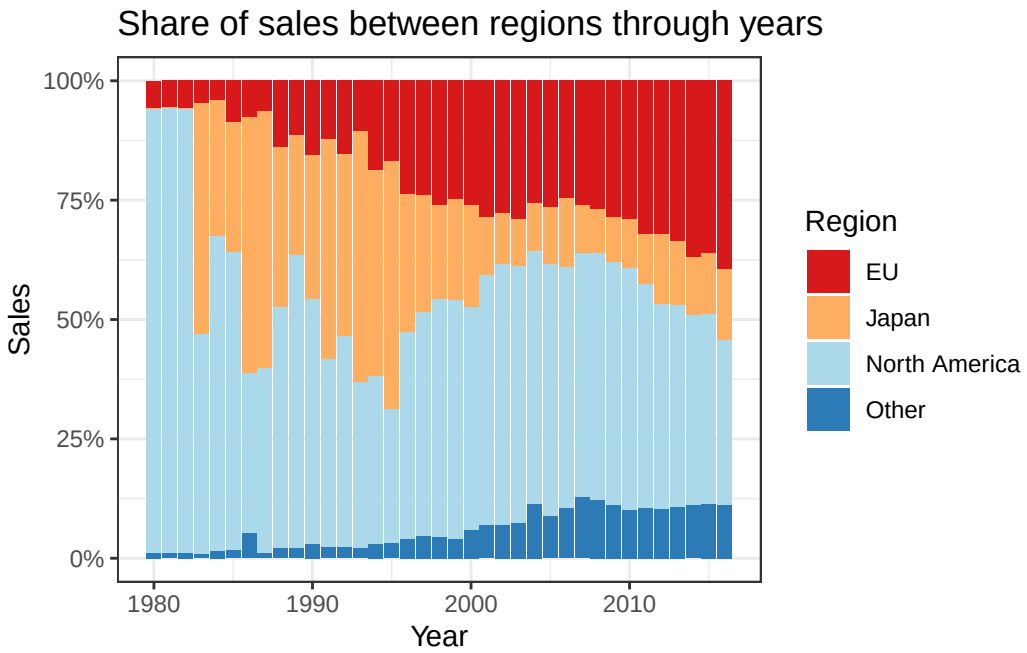

Figure 4: Share of sales between regions through years

The mid-1990 growth of NA and EU markets might be attributed to rise of popularity of PC games at that time (as seen on third plot on Figure 3), that were slowly taking away previously console-only market. Most of the PC games from that period had very small sales in Japan. This theory can be linked also to Figure 3, as rise of PC games popularity resulted in games being more popular and starting to be produced on mass scale. However, this is only just a theory, as besides pure sales data there are many other factors, like socio-political changes, happening at that period that could contribute to this change.

2.2 Publisher and Developer analysis

Let's take into consideration *Publisher* and *Developer* variables. They are similar in their structure (both categorical) and meaning, so the approach to their analysis and exploration will be also similar. Ideally, we would like to replace their character type not by factor (as there are too many unique values to both variables, as shown in Table 1), but by numerical factor corresponding to each publisher and developer importance and influence on sales.

First of all, let's take a look at the top 10 best selling publishers and developers (on Figure 5 below), when comparing their total sales through all years of analysis. From Table 1 we know that *Developer* variable has almost 40% of *NA*s, so I've removed them from plotting (still remaining in *video_sales* data set) for the sake of plot clarity.

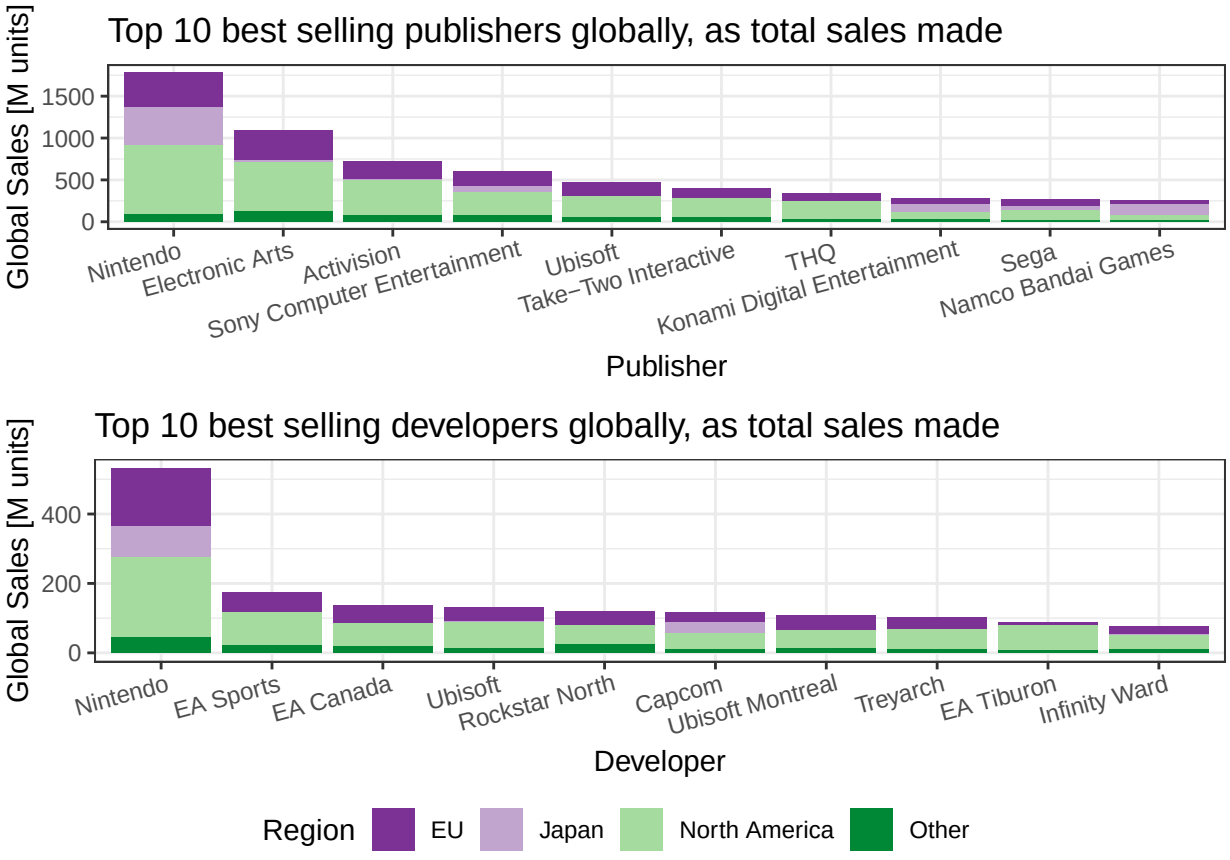

Figure 5: Top 10 best selling publishers and developers, as total sales made

For both variables it's confirming that North America region has the biggest ratio of sales. It is also visible that while EU and Other regions have stable share in all 10 best selling publishers and developers, Japan has clearly playing favorites, favoring ones (like *Nintendo*) and neglecting other names.

It it also visible that most of top 10 places are taken by well known modern corporations. *Nintendo* is both the best selling Publisher, as well as the best selling Developer. Similarly *Ubisoft* and *Electronic Arts/EA* are not far behind. However, seeing as those are big companies, it could be that those companies are releasing many mediocre titles, with only few really successful ones, and their total sum of sales is not guaranteeing that every title they release will be a success. Let's look then at top 10 best selling Publishers and Developers as calculated by sales per title, on Figure 6 below.

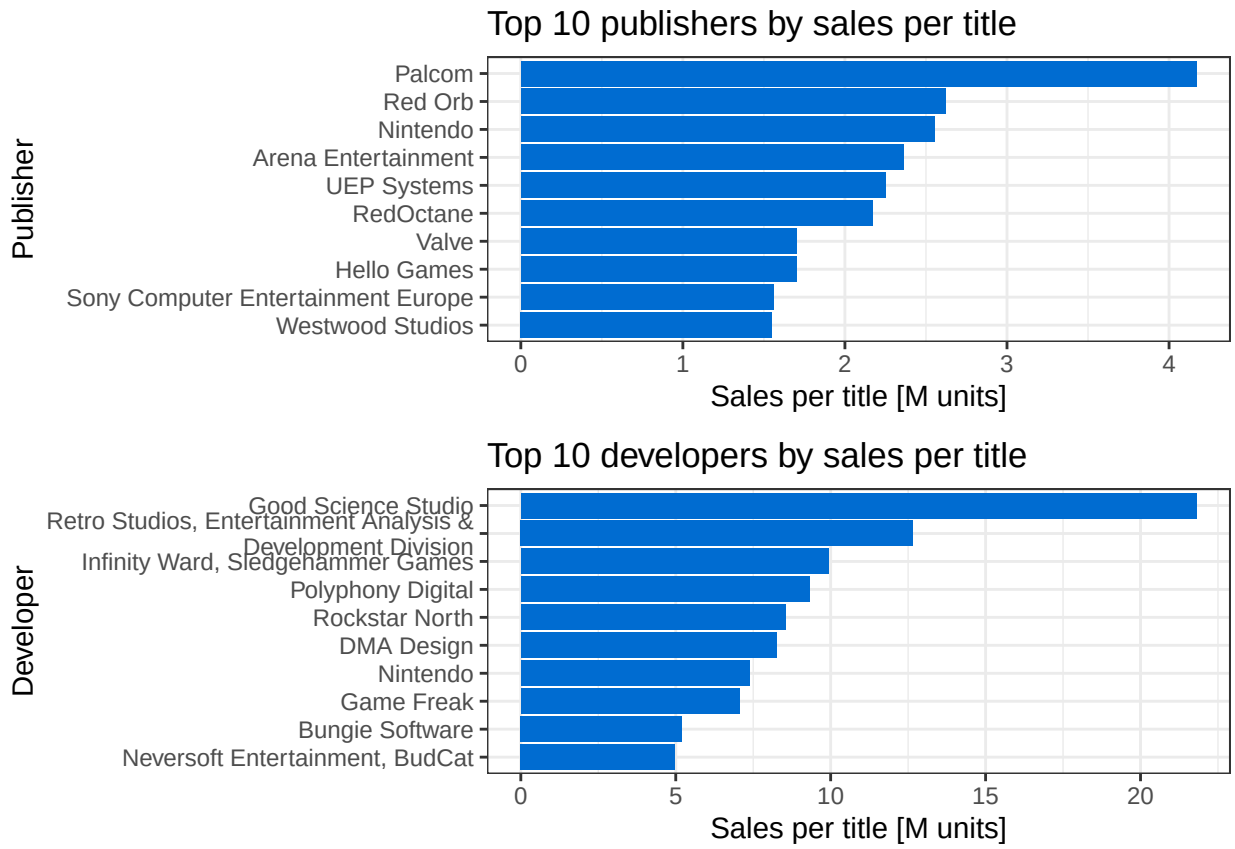

Figure 6: Top 10 publishers and developers as sales per title made

Looking at plots were we consider average sales per title, *Nintendo* and other big players stop being the best selling publishers and developers in our data set. They are falling behind from top places, replaced by not well-known studios (like *Retro Studios, Entertainment Analysis & Development Division* developer studio on second place). This suggest that we should model our sales not by publisher and developer name, but rather on a factors that would be corresponding to sales per title made by each publisher and developer.

Following this approach would however prove unfair advantage on the studios who just released one big title and almost nothing more, and resulting in false fitting in our future models. Let's see how does it look in the top 10 developers and publishers presented on Figure 6.

```
# How many games has top Publisher released?
video_sales %>%
  group_by(Publisher) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n()) %>%
  arrange(desc(sale_per_title)) %>%
  select(Publisher,sale_per_title, no_of_games) %>%  unique() %>% head(10)
```

Table 4: Top 10 best selling publishers

Publisher	sale_per_title	no_of_games
Palcom	4.170000	1
Red Orb	2.620000	2
Nintendo	2.552457	700
Arena Entertainment	2.360000	2
UEP Systems	2.250000	1
RedOctane	2.170000	4
Valve	1.700000	1
Hello Games	1.700000	1
Sony Computer Entertainment Europe	1.558000	15
Westwood Studios	1.550000	1

```
# How many games has top Developer released?
video_sales %>%
  group_by(Developer) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n()) %>%
  arrange(desc(sale_per_title)) %>%
  select(Developer,sale_per_title, no_of_games) %>% unique() %>% head(10)
```

Table 5: Top 10 best selling developers

Developer	sale_per_title	no_of_games
Good Science Studio	21.810000	1
Retro Studios, Entertainment Analysis & Development Division	12.660000	1
Infinity Ward, Sledgehammer Games	9.923333	3
Polyphony Digital	9.314286	7
Rockstar North	8.533571	14
DMA Design	8.260000	2
Nintendo	7.384861	72
Game Freak	7.065000	2
Bungie Software	5.203333	3
Neversoft Entertainment, BudCat	4.980000	1

It is clear from both tables that creating Publisher and Developer factors based just on sales per title would result in favoring a lot of "one-shot" studios. In order to penalize the studios with smaller number of titles released, all sales per title would be multiplied by the function with limit to 1. Good example of such function, which I would use for *Publisher* and *Developer* variables is:

$$f(n) = n * sin(1/n)$$

where n would be the number of titles released. The top selling Publisher and Developer tables with new penalized factors look now like that:

```
# How will publisher factor look like if we penalize studios with small number
# of titles with function with limit to 1?
video_sales %>%
  group_by(Publisher) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n(),
```

```
        adj.factor=no_of_games*sin(1/no_of_games),
        Publisher.fct=sale_per_title*no_of_games*sin(1/no_of_games)) %>%
  arrange(desc(sale_per_title)) %>%
  select(Publisher,sale_per_title, no_of_games, adj.factor, Publisher.fct) %>%
  unique() %>%
  head(10)
```

Table 6: Top 10 best selling publishers with penalized factor

Publisher	sale_per_title	no_of_games	adj.factor	Publisher.fct
Palcom	4.170000	1	0.8414710	3.508934
Red Orb	2.620000	2	0.9588511	2.512190
Nintendo	2.552457	700	0.9999997	2.552456
Arena Entertainment	2.360000	2	0.9588511	2.262888
UEP Systems	2.250000	1	0.8414710	1.893310
RedOctane	2.170000	4	0.9896158	2.147466
Valve	1.700000	1	0.8414710	1.430501
Hello Games	1.700000	1	0.8414710	1.430501
Sony Computer Entertainment Europe	1.558000	15	0.9992594	1.556846
Westwood Studios	1.550000	1	0.8414710	1.304280

```
# How will developer factor look like if we penalize studios with small number
# of titles with function with limit to 1?
video_sales %>%
  group_by(Developer) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n(),
        adj.factor=no_of_games*sin(1/no_of_games),
        Developer.fct=sale_per_title*no_of_games*sin(1/no_of_games)) %>%
  arrange(desc(sale_per_title)) %>%
  select(Developer,sale_per_title, no_of_games, adj.factor, Developer.fct) %>%
  unique() %>%
  head(10)
```

Table 7: Top 10 best selling developers with penalized factor

Developer	sale_per_title	no_of_games	adj.factor	Developer.fct
Good Science Studio	21.810000	1	0.8414710	18.352482
Retro Studios, Entertainment Analysis & Development Division	12.660000	1	0.8414710	10.653023
Infinity Ward, Sledgehammer Games	9.923333	3	0.9815841	9.740586
Polyphony Digital	9.314286	7	0.9966021	9.282637
Rockstar North	8.533571	14	0.9991499	8.526317
DMA Design	8.260000	2	0.9588511	7.920110
Nintendo	7.384861	72	0.9999679	7.384624
Game Freak	7.065000	2	0.9588511	6.774283
Bungie Software	5.203333	3	0.9815841	5.107509
Neversoft Entertainment, BudCat	4.980000	1	0.8414710	4.190525

Now that I've created new numerical factors for *Publisher* and *Developer* variables, we can remove the old character type ones and include the new ones in *video_sales* data set, to be used in future modeling.

```
video_sales <- video_sales %>%
  group_by(Publisher) %>%
  mutate(Publisher.fct=(sum(Global_Sales)/n())*n()*sin(1/n())) %>%
  ungroup()
video_sales <- video_sales %>%
  group_by(Developer) %>%
  mutate(Developer.fct=(sum(Global_Sales)/n())*n()*sin(1/n())) %>%
  ungroup()
video_sales <- video_sales %>%
  select(-Publisher,-Developer)
```

2.3 Platform and Genre analysis

Next variables that I will look into are *Platform* and *Genre*, currently also a character type (see Table 1). Let's first take a look into game titles spread through different types of *Platforms*, on below Figure 7. For the sake of plot clarity, only 15 top Platforms in terms of number of titles released were shown.

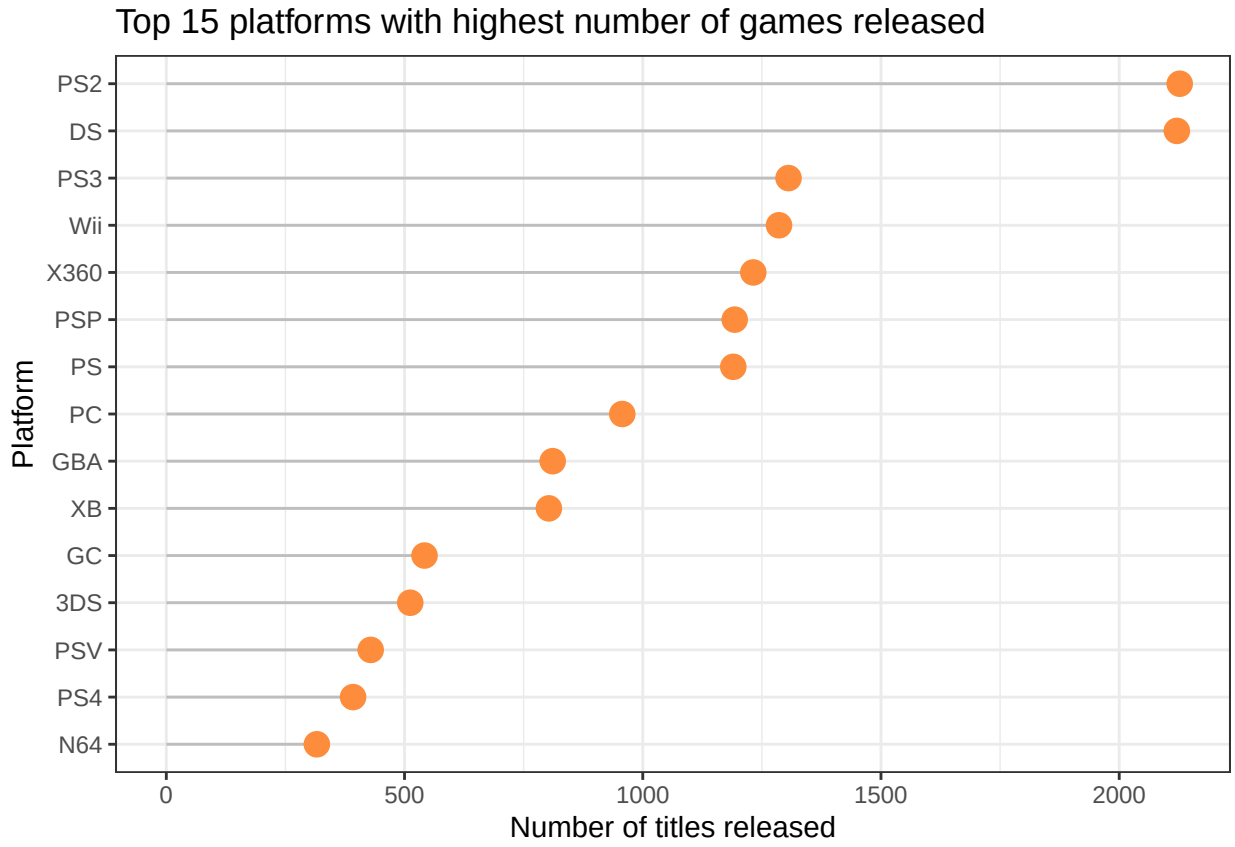

Figure 7: Top 15 platforms with highest number of games released

The indisputable kings, having almost twice as much titles released on them than 3rd place, are *PS2 (PlayStation 2)* console belonging to *Sony* and *DS* console belonging to *Nintendo*. In fact, those two companies own almost all other consoles (12 out of 15) visible on Figure 6, such as *PS3 (PlayStation 3)*, *PSP (PlayStation Portable)*, *PS (PlayStation)*, *PSV (PlayStation Vita)*, *PS4 (PlayStation 4)* belonging to *Sony*, and *Wii*, *GBA*

(Game Boy Advance), *GC (GameCube)*, *3DS*, *N64* belonging to *Nintendo*. The only other *Platforms* from top 15 are *X360 (Xbox 360)* and *XB (Xbox)* belonging to *Microsoft*, and universal *PC* platform.

Platform variable has 31 unique levels, so changing it into a factor may cause overfitting due to too large number of levels (as mentioned in the *Introduction*). Factorizing this variable can improve slightly the speed of future modeling (tested on the current code with 4%=5min improvement), but it also caused drop of fit (higher RMSE and lower R-squared) due to noise caused by zero variance variables (due to platforms with very small number of titles). I've decided not to eliminate *Platform* levels with such near-zero variance, in order not to reduce and disturb the data set, and I'll leave the *Platform* variable as a character-type.

Now let's take a look if the top selling games were released on any of the *Platforms* from Figure 7, and if we can distinguish any pattern in terms of games genre, on Figure 8 below.

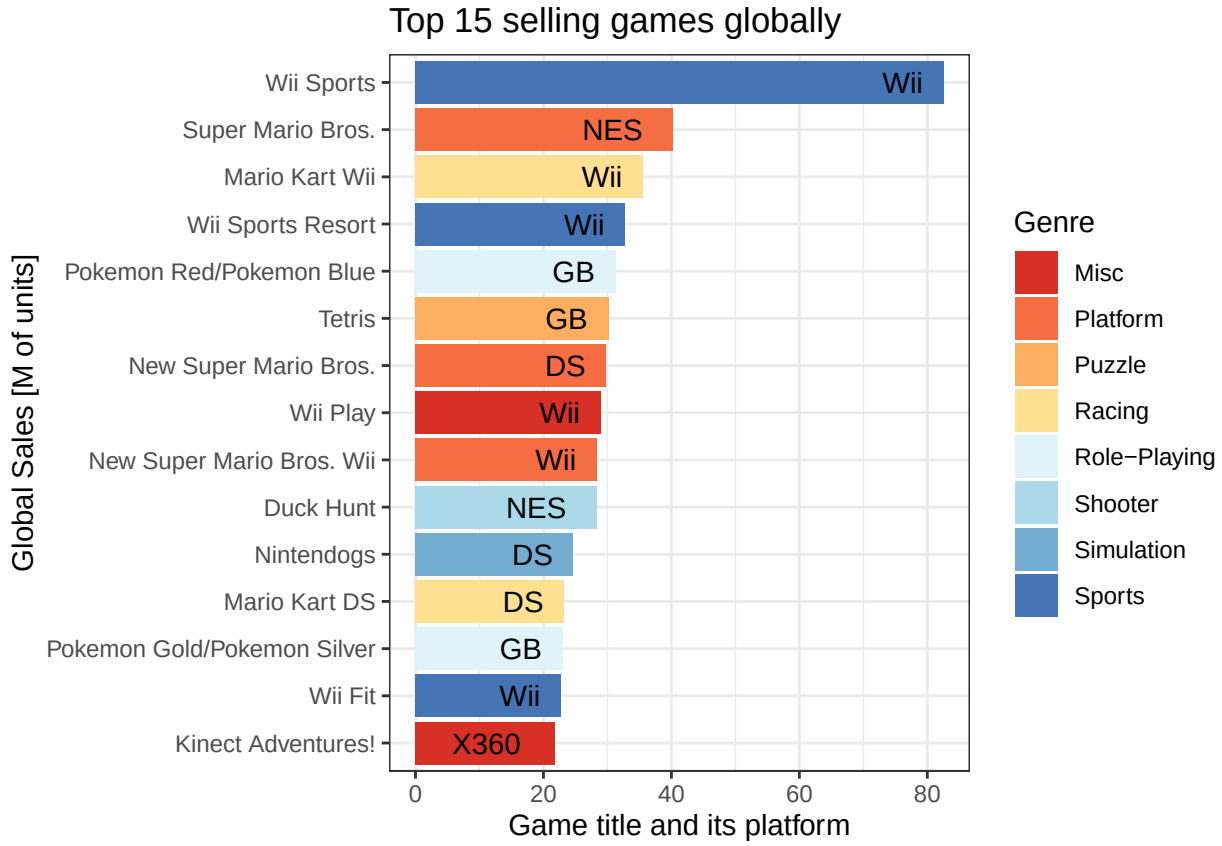

Figure 8: Top 15 selling games globally

It is quite a surprise to see that none of 15 top selling games up until 2016 were released on *PS2*, which was a *Platform* leader, and only 3 were released on *DS*, which was a runner-up. With that we can assume that there is no high correlation between type of Platform the game was released and how well it will sell.

Looking again at Figure 8, it seems that there is also no strong connection between *Genre* of the game and its selling numbers. The *Sport* and *Platform* genres have very small advantage on other titles from the plot, but we're looking here only at the top 15 selling games. The number of titles released might be another influencer.

Let's take a look at the summary of total sales and number of games released of each game genre, in Table 8 below.

```
video_sales %>%
  group_by(Genre) %>%
  summarise(n_titles=n(), total_sales=sum(Global_Sales)) %>%
  arrange(desc(total_sales))
```

Table 8: Summary of total sales and number of games released per genre

Genre	n_titles	total_sales
Action	3307	1717.62
Sports	2306	1310.39
Shooter	1296	1041.83
Role-Playing	1481	930.85
Platform	878	825.86
Misc	1721	790.94
Racing	1226	723.70
Fighting	837	442.63
Simulation	857	388.11
Puzzle	569	240.33
Adventure	1293	233.46
Strategy	673	172.85
NA	2	2.42

First thing that is visible from Table 8 is that there are two titles with *NA* genre. It doesn't make any sense to try to impute them, as *Genre* is clearly categorical variable, and since it's just two titles I will erase them from *video_sales* data set.

```
video_sales <- video_sales %>%
  filter(!is.na(Genre))
```

Since there is only 12 different values of *Genre* variable, and from Table 8 we can see that none of them are represented in very small amount, we can safely assume that transforming *Genre* into factor is the best choice and will not result in overfitting or modeling disruption due to near-zero variances of particular levels.

```
video_sales$Genre <- as.factor(video_sales$Genre)
```

Looking back to Table 8 and comparing numbers of titles per genre and total sales, it seems that it would be good to follow approach of *Publisher* and *Developer* variable, and check the sales per one title per game genre. This is represented on Figure 9 below.

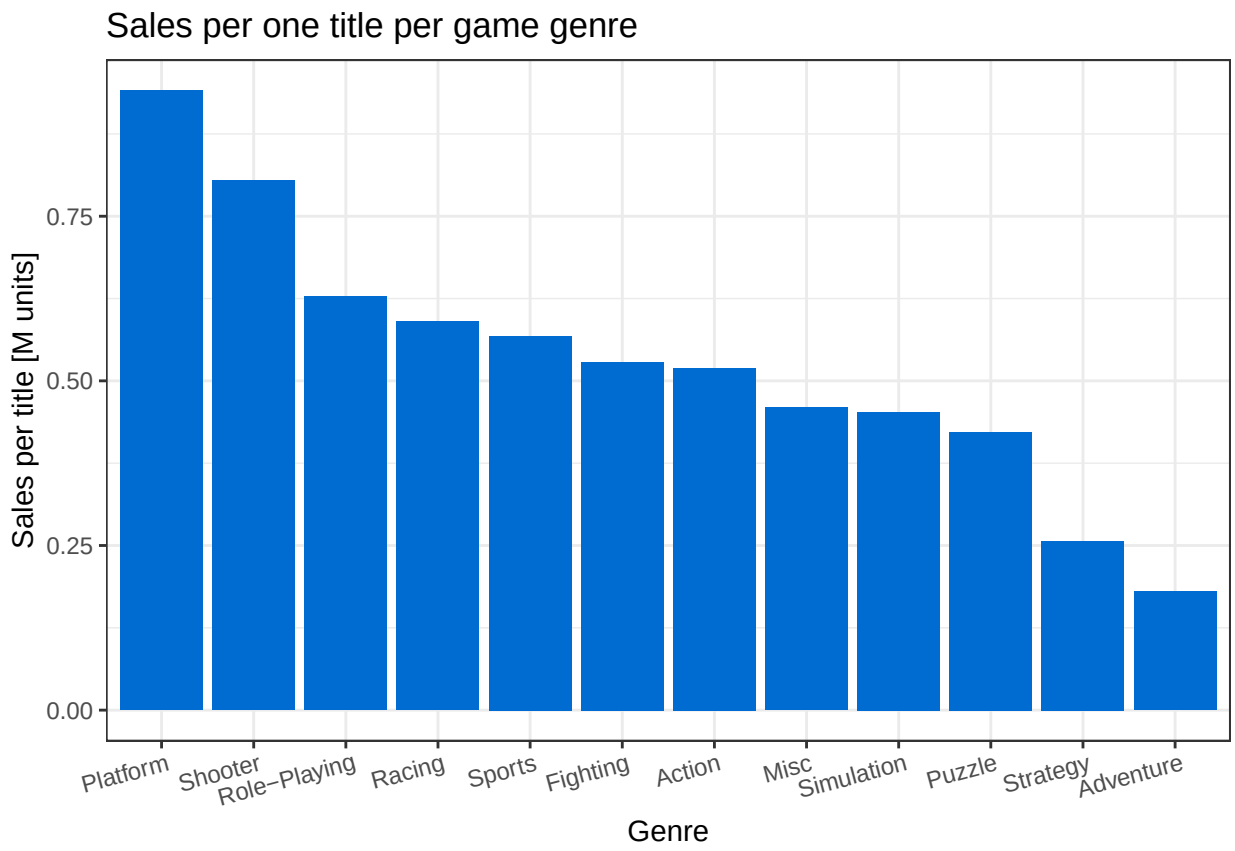

Figure 9: Sales per one title per game genre

It seems that for some genres, considering number of titles released makes a small difference, like for *Platform* genre. However, for many of them they are placed almost on the same level as in Table 8, like *Shooter* or *Role-Playing*. Taking that into consideration, and also the fact that *Genre* doesn't have many levels in total, I take a final decision that it would be best to leave it as non-adjusted factor than change it to numerical representative like for *Publisher* and *Developer*.

2.4 User and Critic Score and Count analysis

Another variables that will be taken into consideration are *User_Score*, *User_Count*, *Critic_Score* and *Critic_Count*. It needs to be noted that if we were to try to predict sales upon game release, or just before it, it would be impossible to know *User*-type variables, as user statistics are created only after the game was released. That's why in this project it will be assumed that we're trying to predict final sales after players (users) had enough time to check the game and give their opinion, as the whole data in the data set was collected also way after release of each game.

First let's take a look at *User_Score* and *Critic_Score* frequencies, on Figure 10 below. Dark background theme was selected for better visibility of very bright colors differentiating different game *Genre* on each plot. As it was pointed in Table 1, both *User_Score* and *Critic_Score* have very high number of *NA*s. For now I will omit them in the visualization.

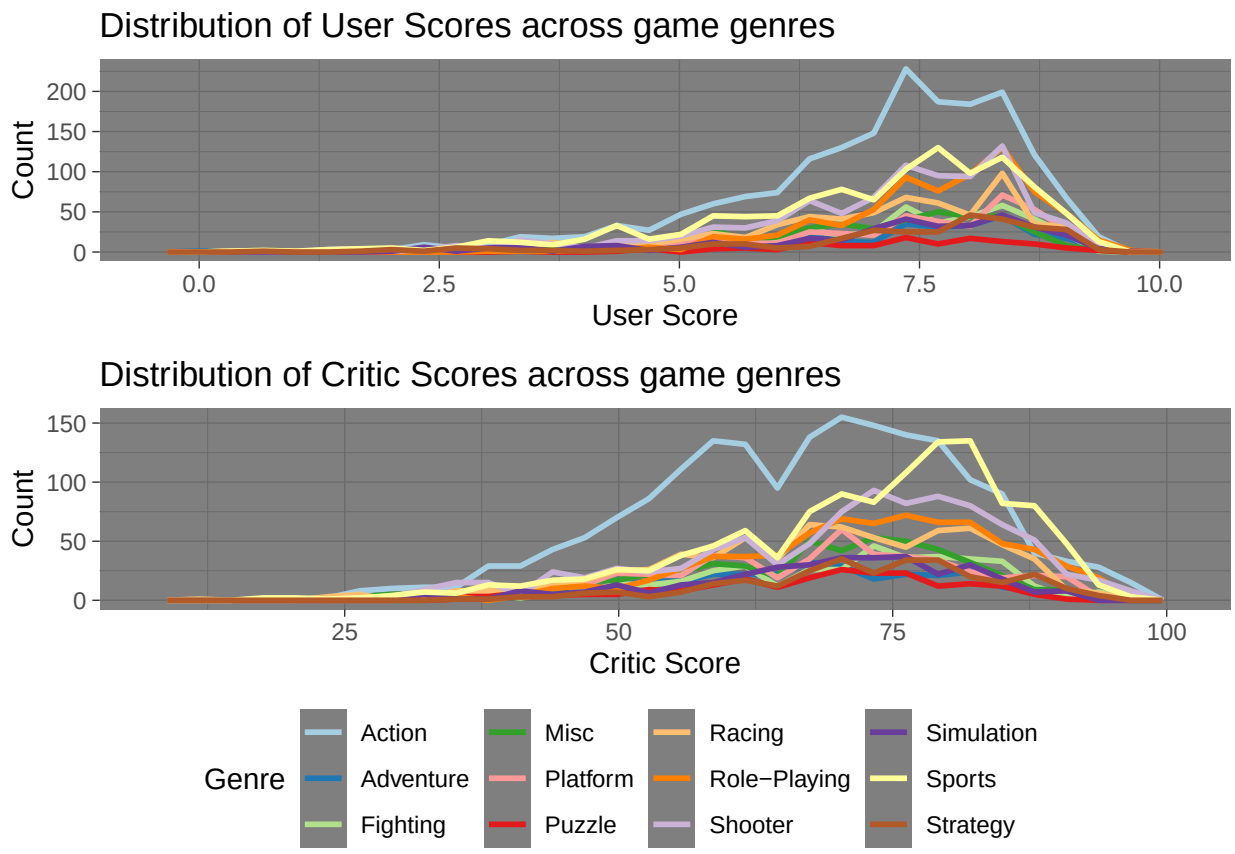

Figure 10: Distribution of User and Critic Scores across game genres

It seems that distribution of User and Critic Scores is quite similar, both having its peak around 80% of max scoring value. Both of the variables have also similar distribution of scoring according to game *Genre*, with highest scores achieved (mostly) by *Action* genre. The only major difference here is *Sports* genre, which Critics favor more than Users do.

As it was mentioned before (and also in Table 1), *User_Score* is a character-type. After removing *NA*s from the first plot on Figure 10, it seems that it's quite obvious that there are no actual character-type values, but they could be skipped due to the nature of the plot. Let's explore this variable further below.

```
video_sales %>%
  group_by(User_Score) %>%
  summarise(n=n()) %>%
  arrange(desc(n)) %>%
  head(10)
```

Table 9: Count of User Scores per type (top 10)

User_Score	n
NA	6605
tbd	2376
7.8	322
8	285
8.2	276
8.3	252
7.5	249
8.5	247
7.9	246
8.1	237

It is immediately clear while looking at Table 9, that *tbd* value (to-be-decided) should actually be *NA*. That would leave only numerical values, so it makes sense to change this variable type to numerical.

```
video_sales$User_Score[video_sales$User_Score=="tbd"] <- NA
video_sales$User_Score <- as.numeric(video_sales$User_Score)
```

Let's see if the same can be said about *Critic_Score* variable, which is also a character-type.

```
video_sales %>%
  group_by(Critic_Score) %>%
  summarise(n=n()) %>%
  arrange(desc(n)) %>%
  head(10)
```

Table 10: Count of Critic Scores per type (top 10)

Critic_Score	n
NA	8461
70	252
71	248
75	240
80	235
73	234
78	233
76	228
72	223
74	223

There are no clearly character-type values, so we can also transform this variable into numeric type.

```
video_sales$Critic_Score <- as.numeric(video_sales$Critic_Score)
```

Scoring coming from Users and Critics seems to have peaks in the same place, as seen on Figure 10. Let's see how they are looking against each other on Figure 11 below. For this plot all *NA* values were skipped.

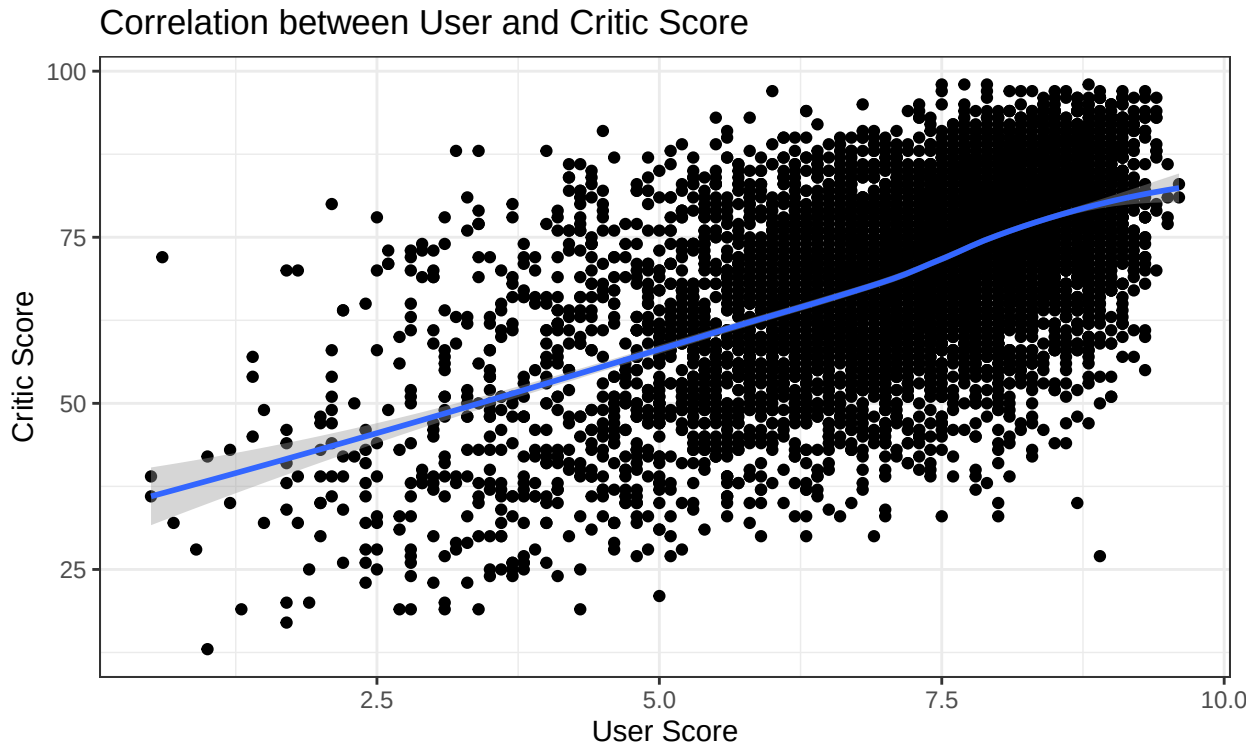

Figure 11: Correlation between User and Critic Score

There seems to be a correlation between *User_Score* and *Critic_Score*, but not very strong one. It is especially visible on the geom_smooth line on Figure 11 (using *loess* method), as the line seems to be slightly wavy and not all straight, indicating that it's not fully linear.

But what about those two variables simultaneous influence on total sales? Figure 12 shows influence of scoring difference between *User_Score* and *Critic_Score* and its impact on *Global_Sales*. Here I also skipped *NA*s from the analysis, and for the sake of comparing like-to-like both scorings were normalized to 100, meaning that all values of *User_Score* had to be multiplied by 10.

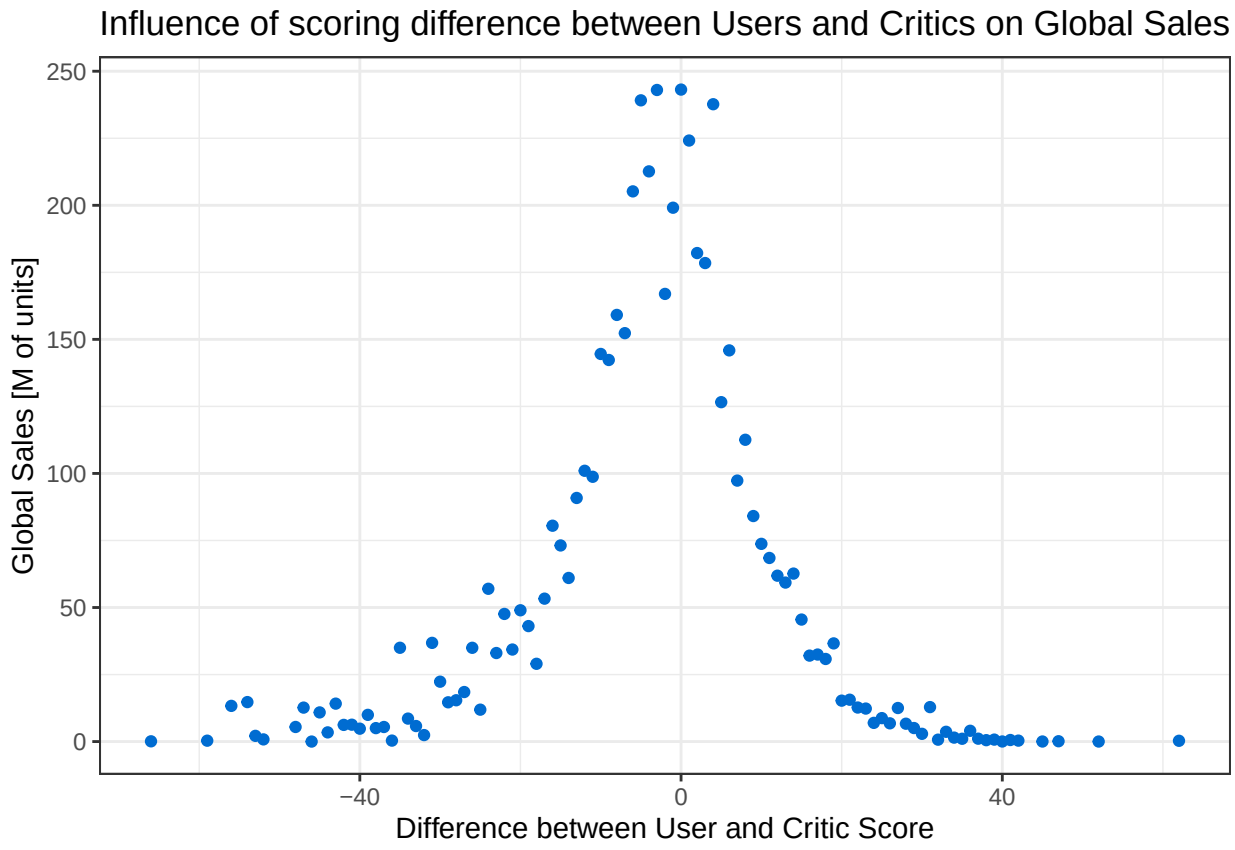

Figure 12: Influence of scoring difference between Users and Critics on Global Sales

Figure 12 shows that scorings towards each other follow normal distribution, with the highest output (*Global_Sales*) achieved when both scorings are exactly the same, and lowest when they are different from each other by at least 30%.

Knowing that both variables should be in agreement for the highest output is not answering the question of at what values of score the sales would be the highest. This question is explored o Figure 13 below, once again omitting *NA*s and normalizing scoring to 100% for visualization purpose.

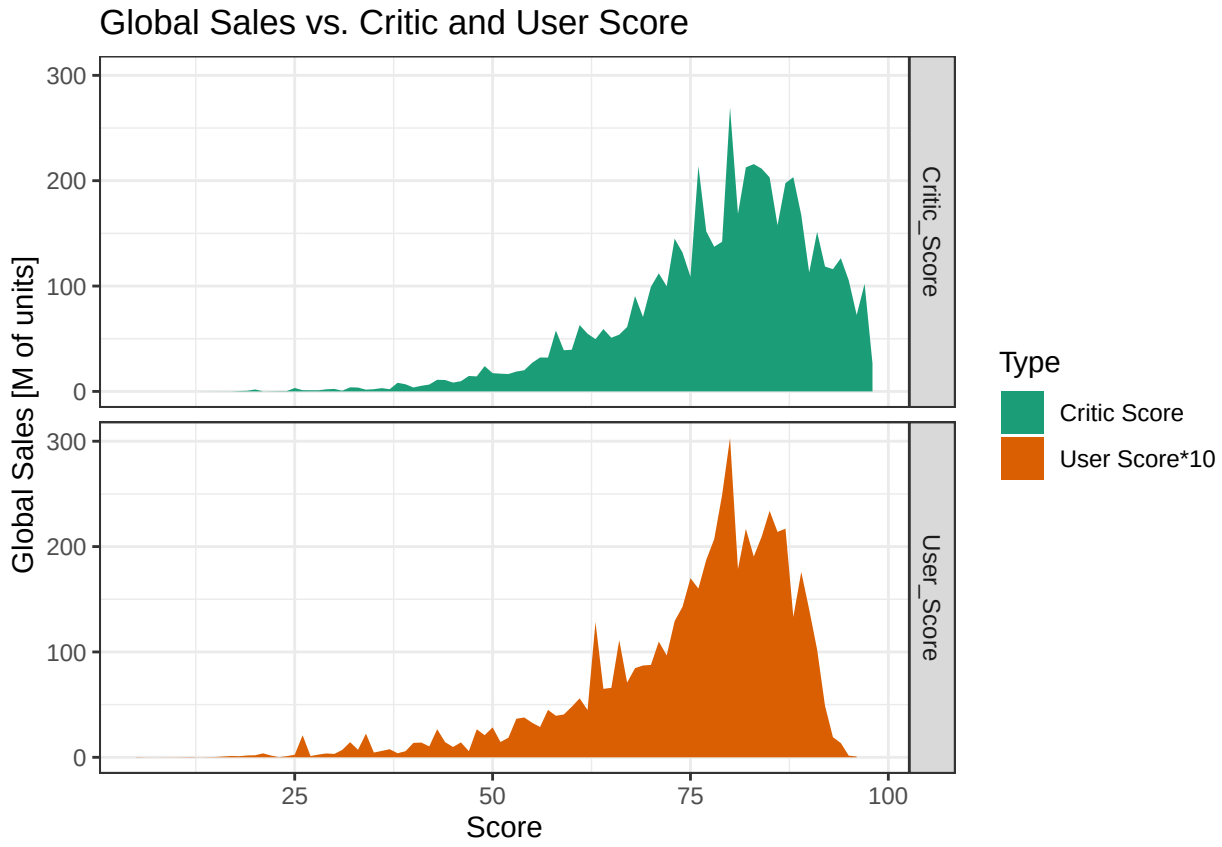

Figure 13: Global Sales vs. Critic and User Score

What is interesting from Figure 13 is that the games that were selling highest number of units were not the games that achieved highest possible scoring from both Users and Critics. Both input sources have similar impact on sales, and the games that were the best sellers were actually the ones that achieved 75-80% of possible score. It seems that even if you're making a masterpiece you still can't please everyone.

Now let's look at the count of scores that were cast. Where their number also influencing *Global_Sales*? Where there any tendencies - the more the better or less is better? This is explored on Figure 14. *NA*s were also skipped on this plot, but since there is naturally more users (players) than critics, the x-scale could not be the same.

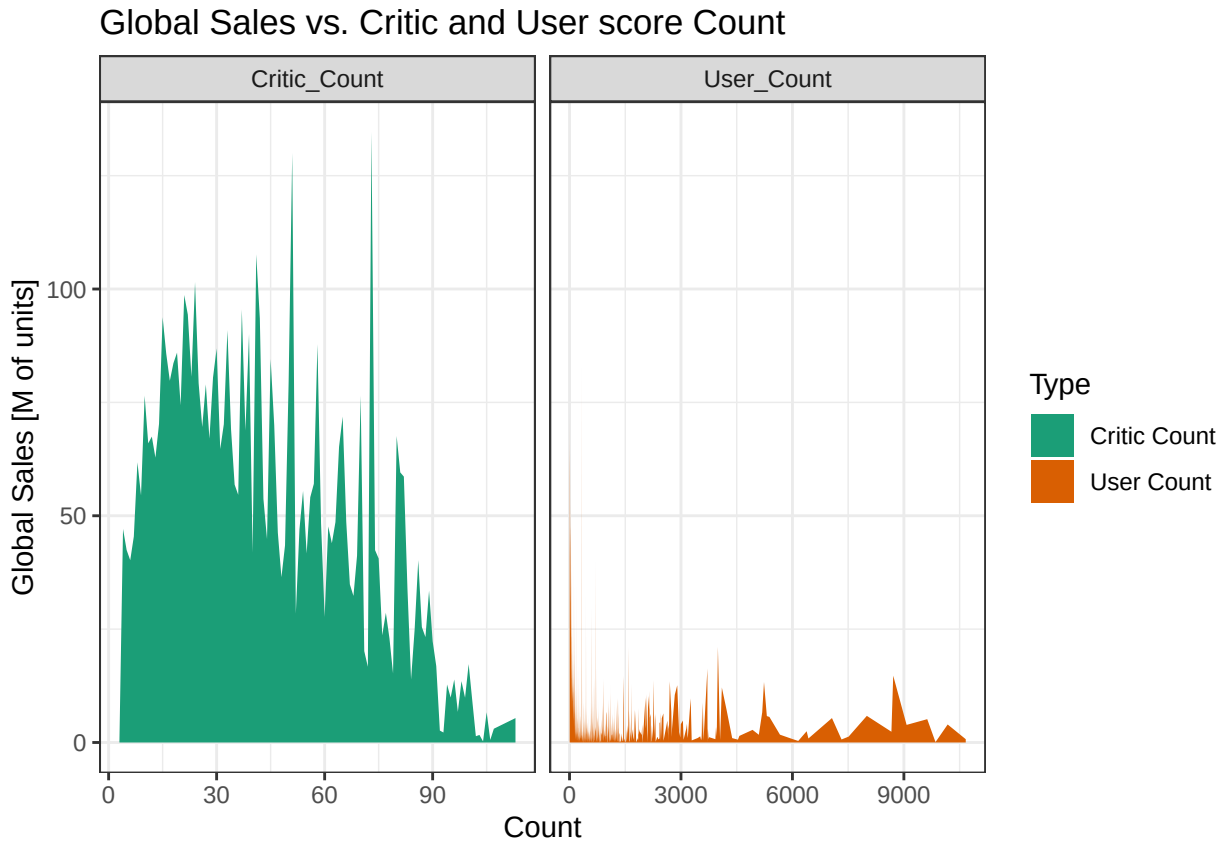

Figure 14: Global Sales vs. Critic and User score Count

There seems to be no relation between two Counts and no clear connection between those two variables and *Global_Sales*. *User_Count* has no clear big peak, and it's influence is spread across many values. Meanwhile *Critic_Count* is clearly right-skewed, with wide-spread peak near lower count values (around 23), and few very singular peaks (similar to whole *User_Count* graph) around higher values.

Since we were looking at correlation of all four User and Critic variables towards *Global_Sales* we now have a pretty good picture of those. But how does the distribution of those values look like across years documented in the data set? This can be seen on Figure 15 below. Like previously, *NA*s were skipped on plots, and scores for Users and Critics are shown as average of all scores per each year, normalized to 100.

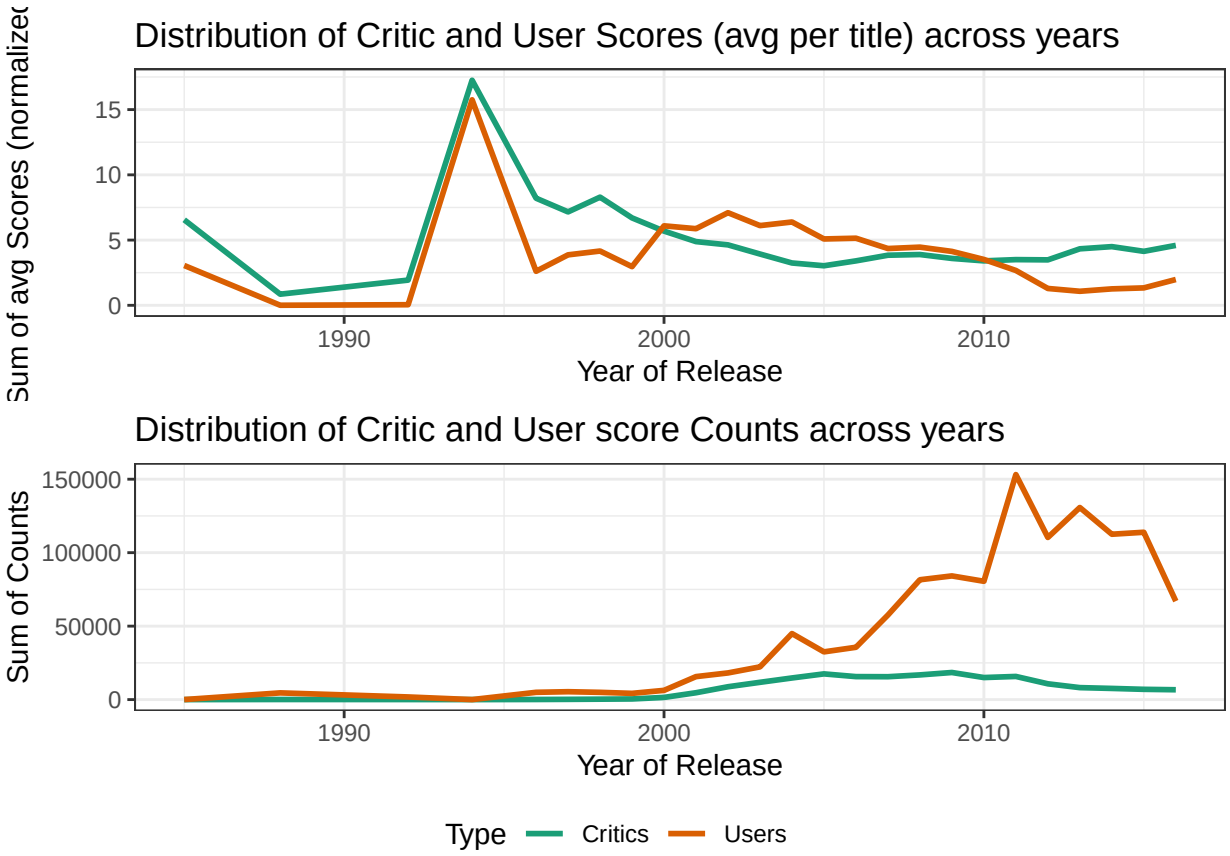

Figure 15: Distribution of Critic and User Scores and Counts across years

Looking at the average scores cast each year (first plot on Figure 15), there is no clear trend across the years. There has been a small decrease in scores around 1990, with (again) big peak near mid-1990s, but since then both scoring in average seem to be on similar level. This is interesting, seeing as the count of scores cast is increasing since that period. It needs to be noted that average score per year is never exceeding 18, and seeing as this is normalized scale up to 100, it is very low. It signals that there must be a lot of pretty badly scored games, and only a few ones are being above-average titles. This trend seems to continue even with increase of number of titles released in recent years (as seen on Figure 3).

As for the counts on Figure 15 (second plot), we can again see a trend starting right after mid-1990s. It could be caused by popularizing gaming industry in total, but in case of User and Critic Scores it is clearly contributed to *Metacritic* portal (which is one of the most popular platform gathering this kind of scores), which started in 1999. Since then we see a drastically big increase of number of scores. Games released previous to 1999 are also having some scoring, as portal was gathering data retroactively and Users can score back, but it's clear there is much less of them.

Since we've noticed there is a significantly less counted scores before mid-1990s, let's check if there are any games that are not scored either from Users or Critics, and are disturbing our data set.

```
video_sales %>%
  filter(User_Count==0 | Critic_Count==0) %>%
  select(Name,Year_of_Release,Global_Sales,User_Score,User_Count,
         Critic_Score,Critic_Count)
```

```
## # A tibble: 0 x 7
## # ... with 7 variables: Name <chr>, Year_of_Release <dbl>, Global_Sales <dbl>,
## #   User_Score <dbl>, User_Count <int>, Critic_Score <dbl>, Critic_Count <int>
```

There are no titles like that, so we can assume that for this context our data set is pure (although it has a title that has been scored 0 - pretty bad game it seems).

Now that we've analyzed all four variables of User and Critic Scores and Count, let's deal with multiple *NA*s that are disturbing them. We can't leave them in, as they would cause errors in several types of machine learning methods, like *Random Forest*, so they need to be dealt with. As mentioned before, removing the titles with them would mean significantly (~50%) decreasing our data set, so the alternative is to impute them with another value. Looking at distribution analysis (like on Figure 10) it seems that both Critic and User values have similar distributions, and inside each other they have clear peaks. Hence it makes sense to impute *User_Score*, *User_Count*, *Critic_Score* and *Critic_Count NA*s with medians of those variables.

```
video_sales$User_Score[is.na(video_sales$User_Score)] <-
  median(video_sales$User_Score, na.rm = TRUE)
video_sales$User_Count[is.na(video_sales$User_Count)] <-
  median(video_sales$User_Count, na.rm = TRUE)
video_sales$Critic_Score[is.na(video_sales$Critic_Score)] <-
  median(video_sales$Critic_Score, na.rm = TRUE)
video_sales$Critic_Count[is.na(video_sales$Critic_Count)] <-
  median(video_sales$Critic_Count, na.rm = TRUE)
```

2.5 Rating variable

Next one is *Rating* variable. As seen before in Table 1 *Rating* has significant number of *NA* values. It's shown currently as character and has 8 unique values in total. Let's see what are those in the Table 11 below.

```
video_sales %>% group_by(Rating) %>% summarise(n=n()) %>% arrange(desc(n))
```

Table 11: Unique values of Rating variable

Rating	n
NA	6676
E	3921
T	2905
M	1536
E10+	1393
EC	8
K-A	3
AO	1
RP	1

Rating system in North America is governed and assigned by ESRB. According to 2016 standards all of the values presented in Table 11 exist (not counting *NA* of course), besides *K-A* and *RP*. *K-A* rating is an old one, replaced by *E* rating, and *RP* means *Rating Pending*, so equal to *NA* in the data set. Let's change the values of *Rating* accordingly.

```
video_sales$Rating[video_sales$Rating=="RP"] <- NA
video_sales$Rating[video_sales$Rating=="K-A"] <- "E"
```

Now, *Rating* variable should have 6 unique values (*NA* not counted), so let's change it to factor for ease of modeling.

```
video_sales$Rating <- as.factor(video_sales$Rating)
```

Similar to what I did before to other variables, let's explore what would be the influence of game *Rating* on total sales per one title released, grouped by each *Rating*. This can be seen on Figure 16 below (*NA*s have been omitted).

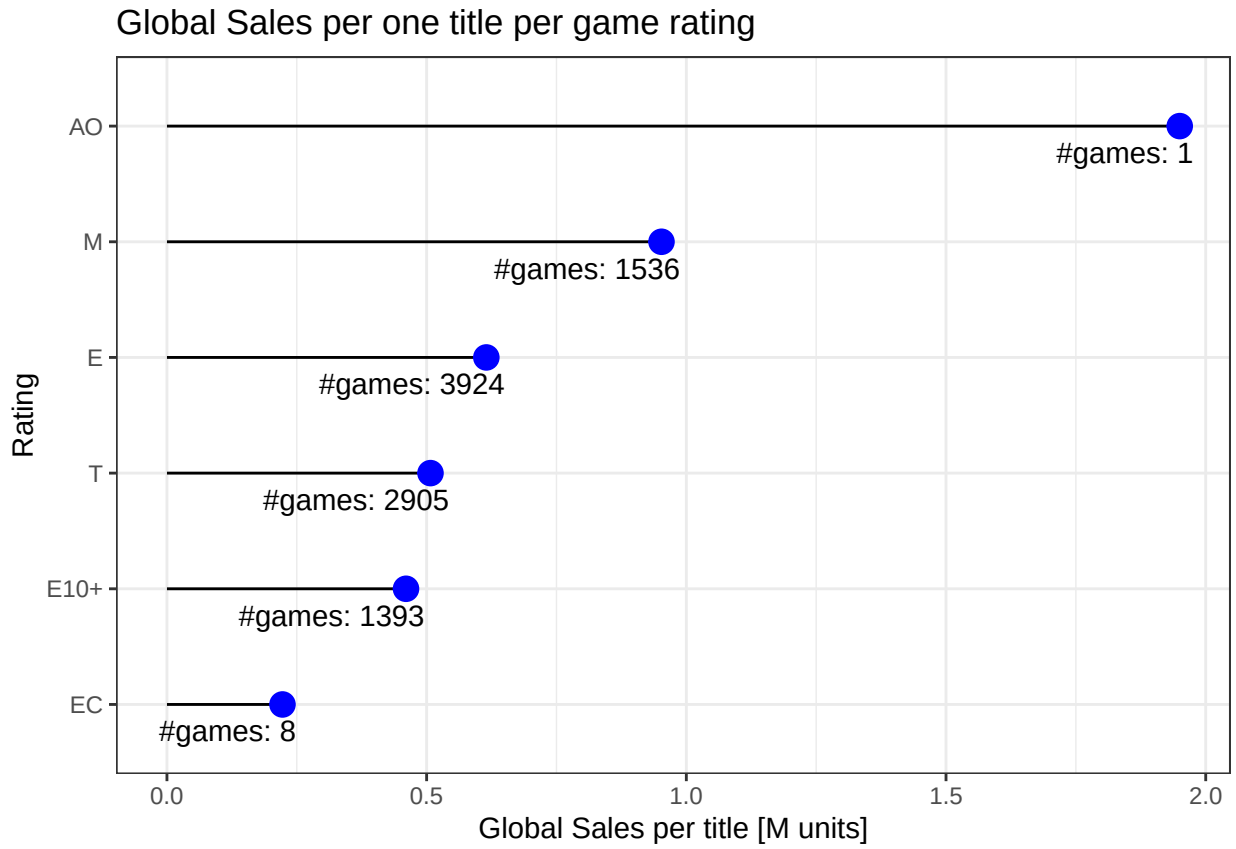

Figure 16: Global Sales per one title per game rating

If we discard the outlier values from Figure 16, which would be Rating *AO* and Rating *EC*, as they have the smallest number of titles released, there is a small advantage of *M* (Mature) *Rating* over the rest in terms of influence on sales. However, all of those ratings are not exceeding 1M units sold per game title, which can be considered low if we compare it to top selling games from Table 2.

The *Rating* variable has significant (41% according to Table 1) number of *NA* values, which would be disturbing our future modeling, as said before. Seeing as the *Rating* variable is clearly a categorical (and

now a factor) variable, we can't impute it with median value. Looking also at the distribution of number of games per rating on Figure 16, we don't see any clear "obvious-choice" to fill *NA*s with. That's why, even if it's reducing the data set substantially, I've decided to exclude rows from the data set where *Rating* is *NA*.

```
video_sales <- video_sales %>% filter(!is.na(Rating))
```

2.6 Game title analysis

The last variable that is left to be analyzed is *Name* variable. As game titles are (usually) belonging to one game only and never repeating, it seems like we should automatically discard this variable from machine learning models. But is there any pattern in titles? If yes, what would be this pattern's influence on sales output?

To help with those questions, I want to create a wordcloud of most used words in the game titles. Using *wordcloud* package, I first create a corpus out of *Name* variable called *text*, that can be used in the actual function.

```
text <- Corpus(VectorSource(video_sales$Name))
```

After that, I clean *text* from numbers, punctuation, unnecessary spaces and stopwords common for English language.

```
text <- text %>%
  tm_map(removeNumbers) %>%
  tm_map(removePunctuation) %>%
  tm_map(stripWhitespace) %>%
  tm_map(removeWords, stopwords("english"))
```

Next, I change cleaned *text* into matrix format representing term document matrix (appearance of words in cleaned *text* as numbers). Then I sum the instances when each word is appearing and rearrange words in decreasing manner. Finally I create a *text.df* data frame with each word and its frequency.

```
text <- as.matrix(TermDocumentMatrix(text))
words <- sort(rowSums(text),decreasing=TRUE)
text.df <- data.frame(word = names(words),freq=words)
```

Now that I have final data frame for wordcloud graph, let's inspect top words that were calculated.

```
text.df %>% select(freq) %>% head(10)
```

Table 12: Top 10 most frequent words in game titles

	freq
the	1161
world	234
star	163
wars	151
nba	151
game	148
lego	138
soccer	136
pro	129
nfl	129

Looking at Table 12 most of the top words seem to be valid, except top one. Since "the" is used in game titles mostly as an article, I would like for it to not be counted in the analysis. As it was not removed automatically during previous cleaning of *text*, I will exclude it manually from *text.df* data frame.

```
text.df <- text.df[!text.df$word=="the",]
```

Other than that, it is also visible from Table 12 that only *world* word is popping out as high frequency word, and the rest of top 20 has pretty similar count.

Now that the data frame created from chopped text string of *Name* is prepared, I can create actual wordcloud below.

```
wordcloud(text.df$word, text.df$freq, min.freq = 50, max.words=200,
          random.order=FALSE, max.freq=400, rot.per=0.35,
          colors=brewer.pal(12, "Paired"))
```

Figure 17: Most used words in game titles

Knowing frequency of each word in game titles, let's try to see how (and if) they are influencing *Global_Sales*.

To find if there is any correlation between those, first I create an additional column called *sales*, that shows sum of *Global_Sales* of all games that had the particular word in the title. To speed up the code I've created a *c* vector beforehand.

```
c <- count(text.df)[[1]]
for (i in 1:c) {
  text.df$sales[i] <-
    video_sales$Global_Sales[str_detect(video_sales$Name,
                                        fixed(text.df$word[i],
                                              ignore_case=TRUE))] %>%
    sum()
}
```

Now that I've changed *text.df* data frame, I can create a correlation plot between sum of *Global_Sales* and frequency of words. It is depicted, together with line smoothed with *loess* method, on Figure 18 below.

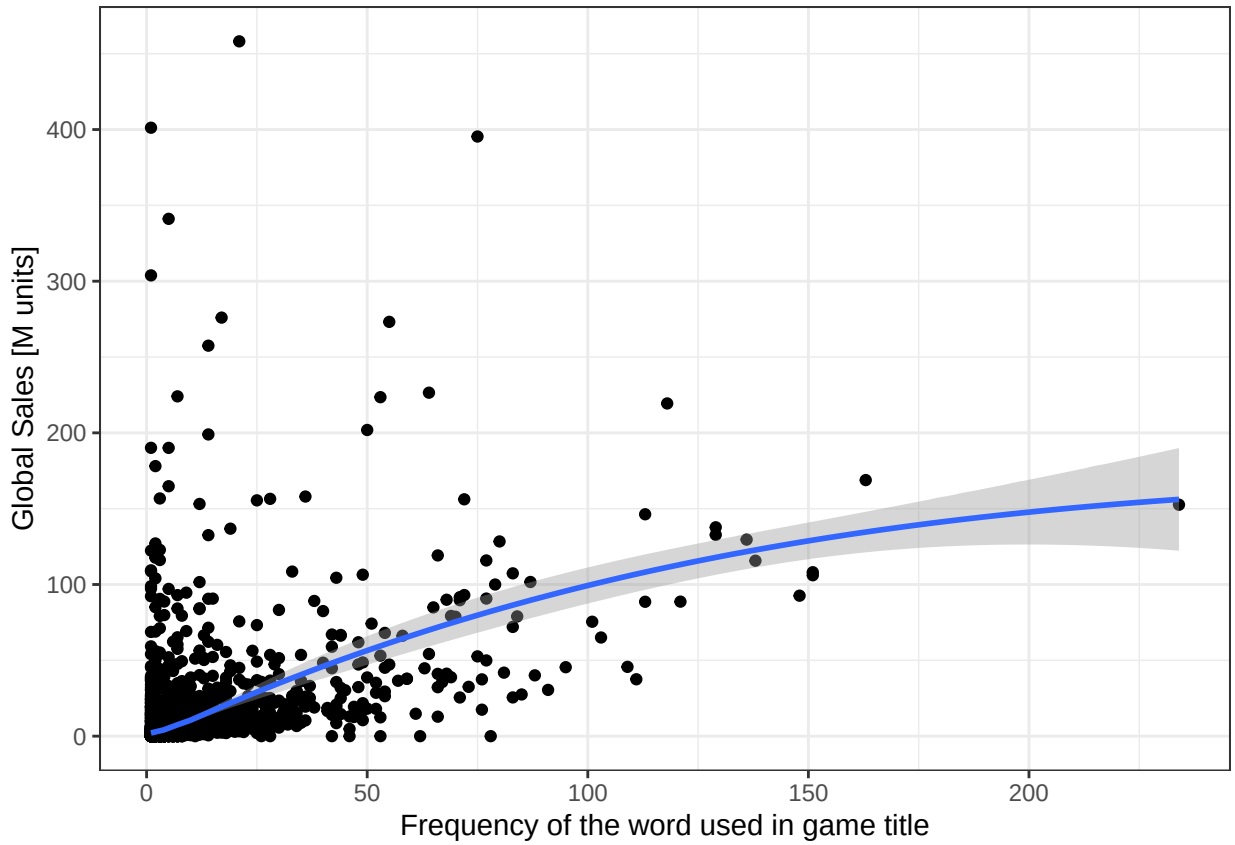

Figure 18: Sales per frequency of words used in the game title

Looking at Figure 18 it doesn't seem like there would be a very strong correlation between words used and *Global_Sales* output (the smoothed line is definitely curving). Due to that I decide that *Name* variable can be removed from *video_sales* data set without replacing it with adjusted word factor, and without hurting significantly the performance of prediction models.

```
video_sales <- video_sales %>% select(-Name)
```

2.7 Final check of the data set

Now, after cleaning and transforming *video_sales* data set, let's explore one more time how the summary of it looks like.

```
summary(video_sales)
```

```
##    Platform         Year_of_Release          Genre         NA_Sales
##  Length:9767        Min.   :1985    Action      :2146   Min.   : 0.000
##  Class :character   1st Qu.:2004    Sports      :1478   1st Qu.: 0.050
##  Mode  :character   Median :2008    Shooter     : 996   Median : 0.120
##                     Mean   :2008    Misc        : 862   Mean   : 0.319
##                     3rd Qu.:2010    Racing      : 856   3rd Qu.: 0.300
##                     Max.   :2016    Role-Playing: 770   Max.   :41.360
```

```
##                                    (Other)     :2659
##     EU_Sales          JP_Sales        Other_Sales        Global_Sales
##  Min.   : 0.0000   Min.   :0.00000   Min.   : 0.00000   Min.   : 0.0100
##  1st Qu.: 0.0100   1st Qu.:0.00000   1st Qu.: 0.01000   1st Qu.: 0.0800
##  Median : 0.0400   Median :0.00000   Median : 0.02000   Median : 0.2100
##  Mean   : 0.1833   Mean   :0.04647   Mean   : 0.06478   Mean   : 0.6138
##  3rd Qu.: 0.1500   3rd Qu.:0.00000   3rd Qu.: 0.05000   3rd Qu.: 0.5700
##  Max.   :28.9600   Max.   :6.50000   Max.   :10.57000   Max.   :82.5300
##
##   Critic_Score    Critic_Count      User_Score      User_Count       Rating
##  Min.   :13.00   Min.   :  3.00   Min.   :0.000   Min.   :    4.0   AO  :   1
##  1st Qu.:63.00   1st Qu.: 14.00   1st Qu.:6.800   1st Qu.:   14.0   E   :3924
##  Median :71.00   Median : 22.00   Median :7.500   Median :   24.0   E10+:1393
##  Mean   :69.38   Mean   : 25.65   Mean   :7.219   Mean   :  129.4   EC  :   8
##  3rd Qu.:77.00   3rd Qu.: 32.00   3rd Qu.:8.000   3rd Qu.:   51.0   M   :1536
##  Max.   :98.00   Max.   :113.00   Max.   :9.700   Max.   :10665.0   T   :2905
##
##  Publisher.fct      Developer.fct
##  Min.   :0.008415   Min.   : 0.008415
##  1st Qu.:0.271861   1st Qu.: 0.183679
##  Median :0.491937   Median : 0.380596
##  Mean   :0.566539   Mean   : 0.605975
##  3rd Qu.:0.818489   3rd Qu.: 0.732777
##  Max.   :2.552456   Max.   :18.352482
##
```

It doesn't seem like there are any outliers that we didn't explore, and all variables are in norm. It doesn't seem also like there are any *NA* values left, but let's double-check.

```
is.na(video_sales) %>% sum()
```

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

As thought, there are no *NA*s left that could disturb modeling. A final look into the status of *video_sales* below tells us, that besides *Platform* there are no other character variables. Only regional sales have values equal to 0 (assuming that those titles were not sold in those regions), and one instance of 0 in *User_Score* was already explored and explained as valid. The *video_sales* data set should be ready for modeling in next steps.

```
status(video_sales)
```

```
##           variable q_zeros      p_zeros q_na p_na q_inf p_inf      type unique
## 1         Platform       0 0.0000000000    0    0     0     0 character     17
## 2  Year_of_Release       0 0.0000000000    0    0     0     0   numeric     25
## 3            Genre       0 0.0000000000    0    0     0     0    factor     12
## 4         NA_Sales     760 0.0778130439    0    0     0     0   numeric    353
## 5         EU_Sales    2079 0.2128596294    0    0     0     0   numeric    277
## 6         JP_Sales    7579 0.7759803420    0    0     0     0   numeric    158
## 7      Other_Sales    2340 0.2395822668    0    0     0     0   numeric    145
## 8     Global_Sales       0 0.0000000000    0    0     0     0   numeric    546
## 9     Critic_Score       0 0.0000000000    0    0     0     0   numeric     81
## 10    Critic_Count       0 0.0000000000    0    0     0     0   integer    106
```

```
## 11      User_Score       1 0.0001023856    0    0      0      0   numeric     95
## 12      User_Count       0 0.0000000000    0    0      0      0   integer    877
## 13          Rating       0 0.0000000000    0    0      0      0    factor      6
## 14   Publisher.fct       0 0.0000000000    0    0      0      0   numeric    282
## 15   Developer.fct       0 0.0000000000    0    0      0      0   numeric    941
```

3 Regression models

3.1 Correlation between variables and PCA

Now that the basic data set for modeling is ready, let's explore different options for training set and used formula.

First, let's think about formula that I will be using. The idea is to first start with predicting total sales, so putting *Global_Sales* as an output. Choosing that, obviously I can't use all of regional sales variables (like *NA_Sales* or *JP_Sales*) as they all add to the same value of total *Global_Sales*.

To check if any of the variables left need to be marked as interacting with each other, I will create a correlation plot of all variables in *video_sales*.

```
corrgram(video_sales[,sapply(video_sales, is.numeric)],
         order=TRUE, lower.panel=panel.shade,
         upper.panel=panel.cor, text.panel=panel.txt,
         main="Correlogram of variables selected for modeling")
```

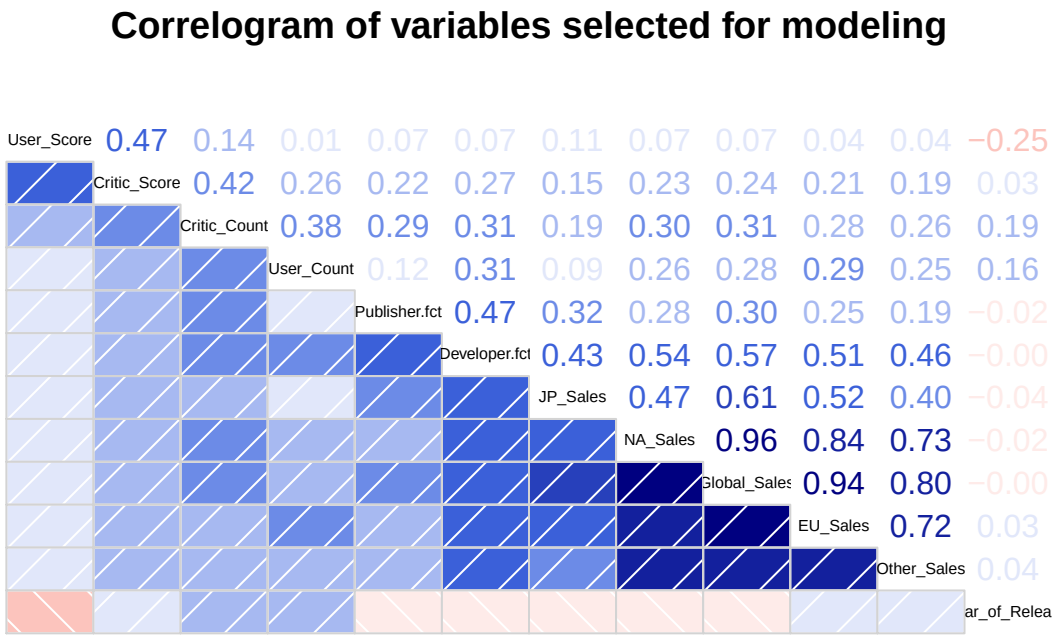

Figure 19: Correlogram of variables selected for modeling

As it's visible on plot above, variables that could be considered cross-depended are most of sales variables, especially *Global_Sales*, *NA_Sales* and *EU_Sales* (with R>0.8). I assume that rest of the variables are independent from each other.

Second thing I wanted to look at before starting modeling was how to define training and test sets for models. From the first glance on the data set (Table 1) and correlation plot above, it doesn't make much sense to follow PCA (Principal Component Analysis), as there are not so many variables and they are not tightly interacting with one another (a lot of variables have R<0.3 between each other). However, I was quite curious to see how many components would be needed to explain most of variance, and check if this still might be better approach than just slicing *video_sales* into training and testing sets.

For now, for the purpose of this PCA I've focused only on numerical variables, creating a *pca_data* out of *video_sales*, keeping in mind that if we would be following this approach all variables would need to be transformed to numerical equivalents.

```
pca_data <- video_sales %>% select(-Platform,-Genre,-Rating)
```

Next, I've calculated actual components split out of *pca_data*.

```
pr.vsales <- prcomp(pca_data, scale=TRUE)
summary(pr.vsales)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5    PC6     PC7
## Standard deviation     2.1940 1.2626 1.1699 1.00588 0.87612 0.7684 0.74195
## Proportion of Variance 0.4012 0.1328 0.1141 0.08432 0.06397 0.0492 0.04587
## Cumulative Proportion  0.4012 0.5340 0.6481 0.73238 0.79634 0.8455 0.89141
##                           PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.65704 0.63452 0.56097 0.39247 0.00293
## Proportion of Variance 0.03597 0.03355 0.02622 0.01284 0.00000
## Cumulative Proportion  0.92739 0.96094 0.98716 1.00000 1.00000
```

Having done that, I've calculated and created plots of variance and cumulative variance of PCA calculated one step before.

```
pr.var <- pr.vsales$sdev^2
pve <- pr.var/sum(pr.var)
```

```
par(mfrow=c(2,1))
plot(pve, type="b", lwd=2, col="#FD8D3C",
     main="Variance of PCA",
     xlab="Number of Principal Components",
     ylab="% of Variance")
plot(cumsum(pve), type="b", lwd=2, col="#41B6C4",
     main="Cumulative variance of PCA",
     xlab="Number of Principal Components",
     ylab="% of Cumulative Variance")
abline(h=0.9, lty=2)
```

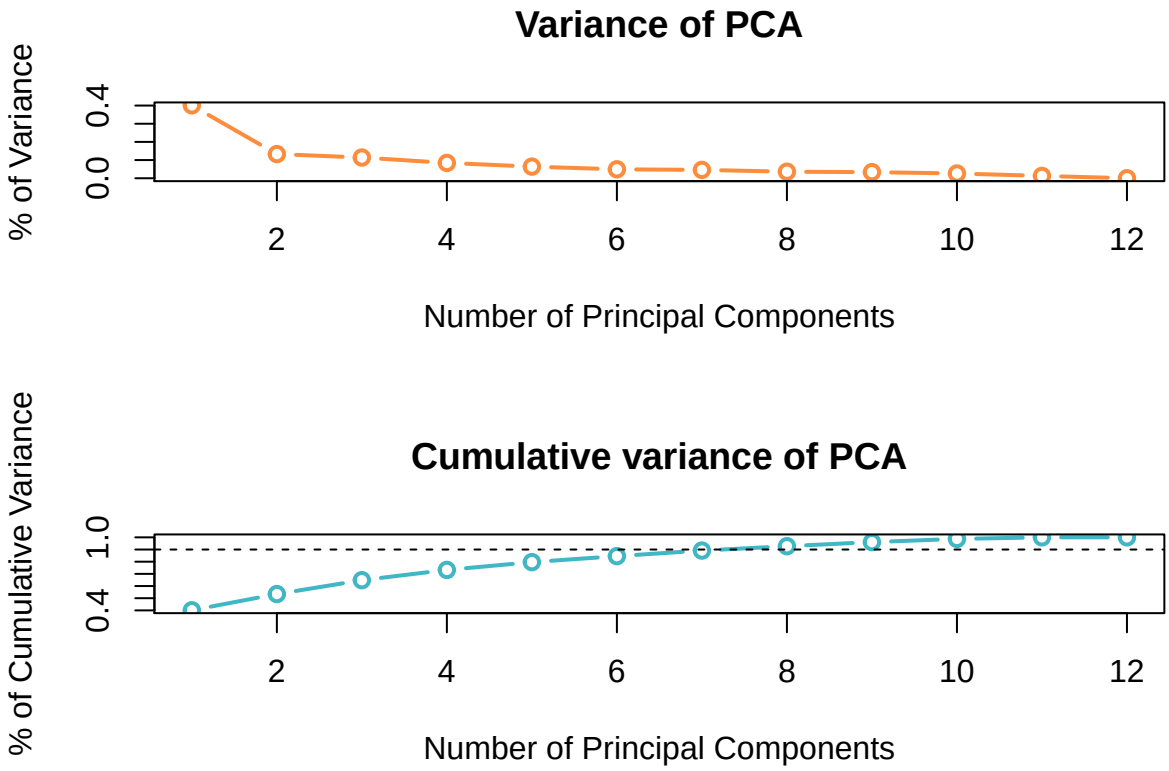

Figure 20: Variance and cumulative variance of PCA

The dashed line on cumulative variance plot above marks moment when components explain 90% of variance. It's evident that we would need 7 components for that, out of 12 (keeping in mind that three variables are missing from this analysis as they were not transformed into numerical ones). This would mean that using PCA method for modeling I would still need at least 60% of whole data set. Given that we don't have as many variables or very big data set in the first place, I've decided to follow classic approach of simple splitting, instead of PCA.

3.2 Predicting global sales

As a first step before training prediction models for *Global_Sales* as an output, I create a new data set out of cleaned *video_sales* one. This one, called *globalsales* will consist only of variables that are used as predictors and the output.

Despite being the most correlated towards *Global_Sales* (see Figure 19 above), regional sales would not be helpful in actual prediction model. The sum of them is adding up to value of *Global_Sales*, so they can't be used all together as they would be self-explaining the output. Using some of them could be an approach that some wants to follow, but I've decided to first build a model based on predictors other than sales. In this case, using even one of the regional sales variables could drown the other predictors.

```
globalsales <- video_sales %>%
  select(-NA_Sales,-EU_Sales,-Other_Sales, -JP_Sales)
```

Now, let's create a train and test sub-sets by splitting *globalsales* following 80/20 Pareto Principle, with 80% of *globalsales* data set selected for training in *global.train* and the rest for testing in *global.test*.

```
global.test_index <- createDataPartition(y = globalsales$Global_Sales,
                                         times = 1, p = 0.2, list = FALSE)
global.test <- globalsales %>% slice(global.test_index)
global.train <- globalsales %>% slice(-global.test_index)
```

As some of the variables are not numerical, let's make sure that their unique representation is present equally in both *global.train* and *global.test* sets.

```
global.train <- global.train %>%
  semi_join(global.test, by = "Platform") %>%
  semi_join(global.test, by="Genre") %>%
  semi_join(global.test, by="Rating")
```

For modeling I've selected five regression models:

- Linear Regression (LM)
- Cubist
- Regression Trees (Rpart)
- Random Forest (RF)
- Non-Negative Least Squares (NNLS)

All of them I'll run with *train* function from *caret* package (for Cubist and NNLS additional packages are required), and create a vector of predicted outputs using *predict* function. All models would be 10-fold cross-validated, to try to generalize as much as possible with different data sets and avoid overfitting. Additionally to that, in all models input variables will be scaled and centered as pre-processing. Models without pre-processing were also tested, but mostly they were performing worse, with higher RMSE and lower R-square results.

Starting from Linear Regression, the code for training and predicting is presented below. Linear Regression function doesn't have any additional parameters, so none could be tuned.

```
global.lm.fit <- train(Global_Sales~.,data=global.train, method="lm",
                       trControl=trainControl(method="cv",number=10),
                       preProcess = c("center", "scale"))

global.lm.pred <- predict(global.lm.fit,global.test)
```

Second model trained was following Cubist method. It was also pre-processed and 10-fold cross-validated. *Global.cubist.fit* fitted model and *global.cubist.pred* prediction vector were created.

```
global.cubist.fit <- train(Global_Sales~.,data=global.train, method="cubist",
                           trControl=trainControl(method="cv",number=10),
                           tuneGrid=data.frame(committees=seq(1,9,1),
                                               neighbors=seq(1,9,1)),
                           preProcess = c("center", "scale"))

global.cubist.pred <- predict(global.cubist.fit,global.test)
```

As Cubist can have *committees* and *neighbors* parameters tuned, I've been trying to find best fit using *tuneGrid* parameter in *train* function. Both tuned parameters can be integers with values less than 10, so

range from 1 to 9 seems to be the simplest one. Plot showing the best fit of those parameters that minimizes RMSE is shown below.

```
plot(global.cubist.fit,highlight = TRUE)
```

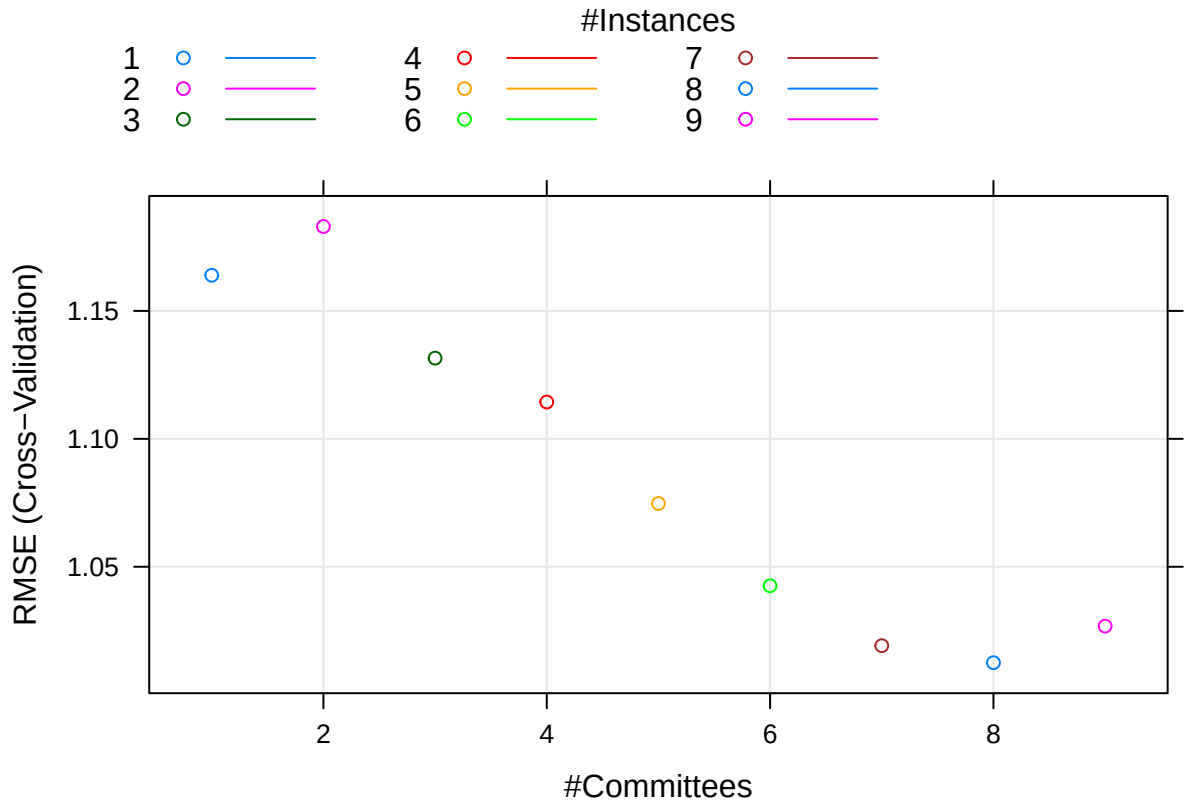

Figure 21: RMSE values for different tuned parameters for Cubist model of Global Sales

The best tune value is therefore:

```
global.cubist.fit$bestTune
```

```
##   committees neighbors
## 8          8         8
```

Third model that I was training was Regression Trees. It was also pre-processed and 10-fold cross-validated. Regression Trees method (represented by *rpart*) has tunable complexity parameter *cp*. The code for model and prediction vector creation is visible below.

```
global.rpart.fit <- train(Global_Sales~.,data=global.train, method="rpart",
                          tuneGrid=data.frame(cp=seq(0,0.05,0.005)),
                          trControl=trainControl(method="cv",number=10),
                          preProcess = c("center", "scale"))

global.rpart.pred <- predict(global.rpart.fit,global.test)
```

As *cp* is also tunable parameter, let's see on the graph at which value we can achieve minimal RMSE.

```
ggplot(global.rpart.fit, highlight=TRUE)
```

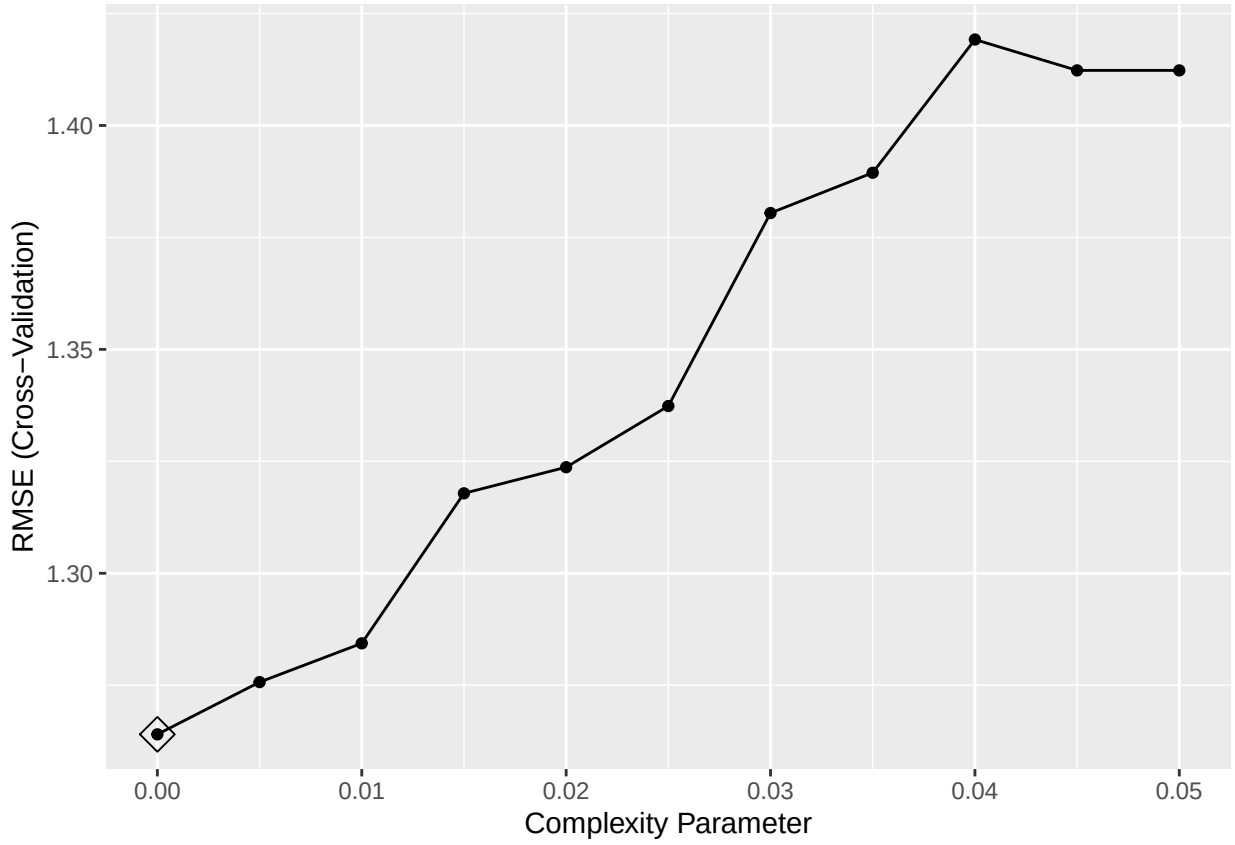

Figure 22: RMSE values for different tuned parameters for Rpart model of Global Sales

```
global.rpart.fit$bestTune
```

```
##   cp
## 1  0
```

The best fit according to code and plot above is for *cp* equal to 0. In Regression Tree method, *cp* equal 0 means that it is better to not prune the tree at all and create as many splits as possible. Due to that I've decided not to plot created tree, as it would be unreadable. It is also first indicator that the fit here might not be very good, or our data set is too small for this prediction.

Fourth model that I was training was Random Forest type. Here, an additional fixed parameter besides 10-fold cross-validation and pre-processing was introduced - number of trees *ntree*. I set is up as a standard 100, as any higher number was heavily influencing calculation time with not significant improvement in results. Tuned parameter here is *mtry*, for which I've tried to find best fit from integers between 1 and 25.

```
global.rf.fit <- train(Global_Sales~., data = global.train, method="rf",
                                    trControl=trainControl(method="cv", number=10),
                                    tuneGrid=data.frame(mtry=seq(1,25,1)),
                                    ntree=100, preProcess = c("center", "scale"))

global.rf.pred <- predict(global.rf.fit,global.test)
```

To see what value of *mtry* is defined as best and minimizing RMSE, I plot the tuned outputs and check within model *bestTune* variable.

```
global.rf.fit$bestTune
```

```
##    mtry
## 24   24
```

```
ggplot(global.rf.fit, highlight=TRUE)
```

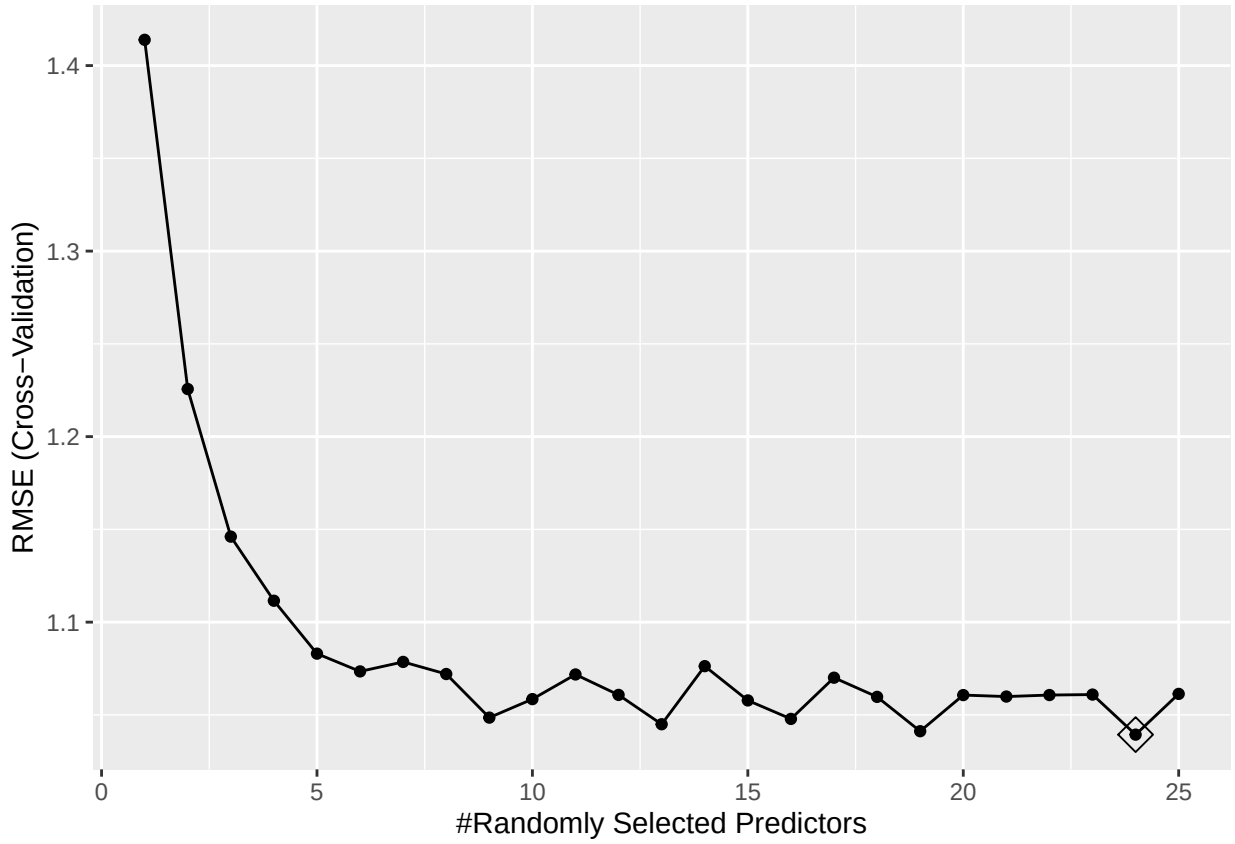

Figure 23: RMSE values for different tuned parameters for RF model of Global Sales

It seams that best value of *mtry* would be around 24, however by looking closely it seems that since 10 the RMSE value is up and down just slightly and overall could be perceived as constant.

Last model that I was training here was Non-Negative Least Squares (also requiring additional *nnls* package for calculations). This model does not have any tunable parameters, so it was only pre-processed and cross-validated, same as previous models.

```
global.nnls.fit <- train(Global_Sales~.,data=global.train, method="nnls",
                         trControl=trainControl(method="cv",number=10),
                         preProcess = c("center", "scale"))

global.nnls.pred <- predict(global.nnls.fit,global.test)
```

Now that all models have been created, let's compare the RMSE and R-squared results of all of them. I've created a list of fitted models, that would be then used as input for resampling.

```
models <- list(lm=global.lm.fit,cubist=global.cubist.fit,rpart=global.rpart.fit,
               rf=global.rf.fit,nnls=global.nnls.fit)
```

This list is then used as input for RMSE, MAE and R-squared boxplots, which we get if we *resample* the list beforehand.

```
bwplot(resamples(models),metric=c("RMSE","Rsquared","MAE"),
       main="Global Sales learning models comparison")
```

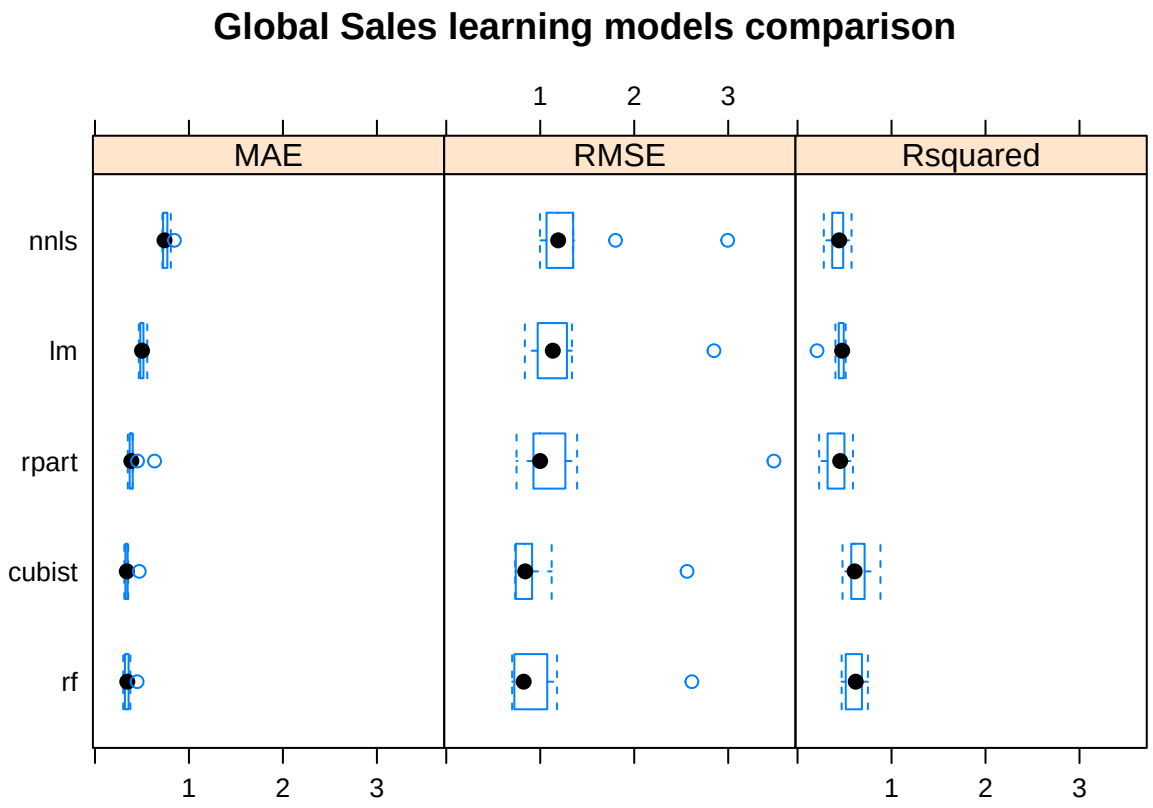

Figure 24: Global Sales learning models comparison

The boxplot above gives us already an indication which models might be performing better than others, but since their results are quite close to one another, it's hard to tell exactly median values for each. That's why I've created a *global.results* data frame below, that lists median values of RMSE, MAE and R-squared of all models from all their runs that have been calculated above.

```
mae <- list(median(global.lm.fit$resample$MAE),
            median(global.cubist.fit$resample$MAE),
            median(global.rpart.fit$resample$MAE),
            median(global.rf.fit$resample$MAE),
            median(global.nnls.fit$resample$MAE))
rmse <- list(median(global.lm.fit$resample$RMSE),
             median(global.cubist.fit$resample$RMSE),
             median(global.rpart.fit$resample$RMSE),
             median(global.rf.fit$resample$RMSE),
             median(global.nnls.fit$resample$RMSE))
rsqr <- list(median(global.lm.fit$resample$Rsquared),
             median(global.cubist.fit$resample$Rsquared),
             median(global.rpart.fit$resample$Rsquared),
             median(global.rf.fit$resample$Rsquared),
             median(global.nnls.fit$resample$Rsquared))

global.results <- data.frame(Model=c("LM","Cubist","Rpart","RandomForest","NNLS"),
                             Median_MAE=unlist(mae), Median_RMSE=unlist(rmse),
                             Median_Rsquared=unlist(rsqr))
```

Table 13: Median MAE, RMSE and R-squared results for Global Sales prediction models

Model	Median_MAE	Median_RMSE	Median_Rsquared
LM	0.5006141	1.1338922	0.4749125
Cubist	0.3384910	0.8401069	0.6077449
Rpart	0.3869503	0.9988440	0.4537697
RandomForest	0.3438944	0.8238146	0.6194075
NNLS	0.7399864	1.1910994	0.4433690

From the table above we can see that none of the selected models are statistically significant, as they all have R<0.65 and quite high RMSE. The closest ones are Cubist and Random Forest (RF).

Let's take a final look on all models outputs (actual and predicted ones) on the figure below.

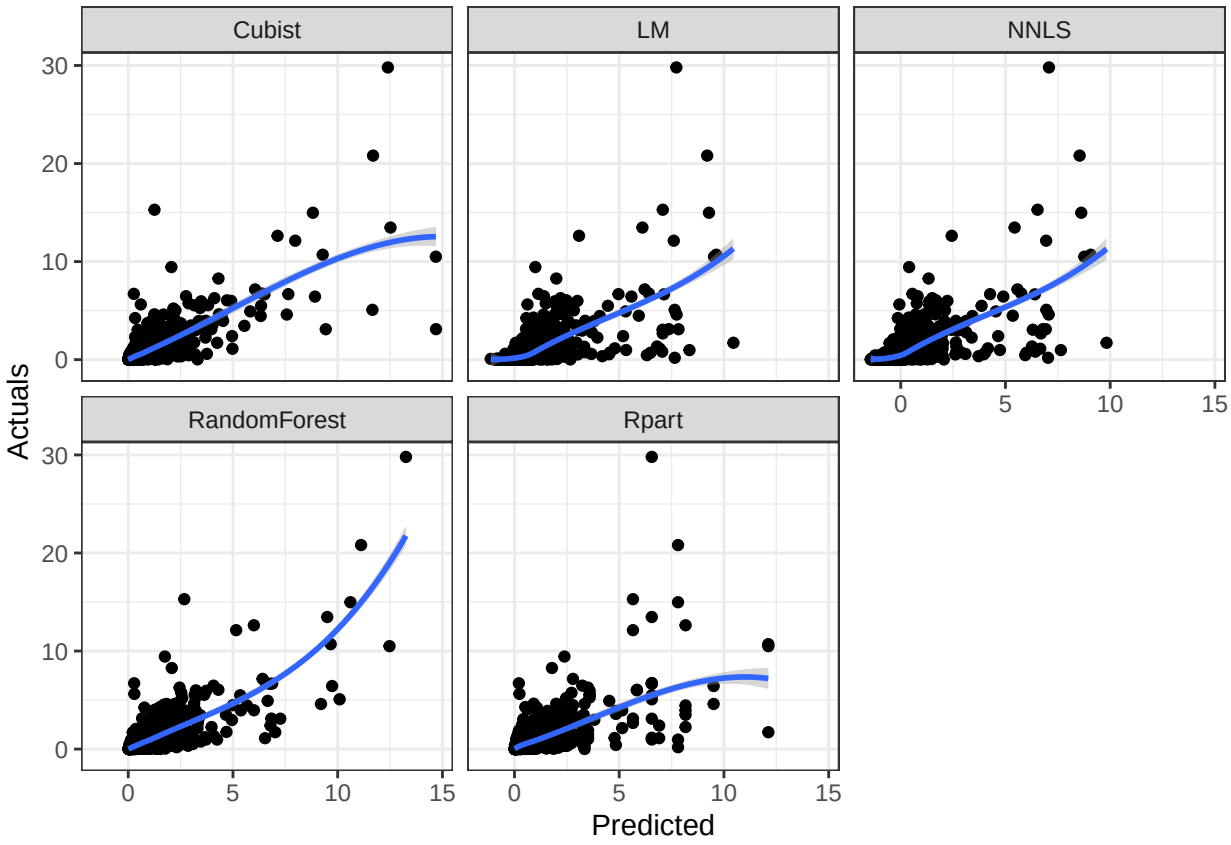

Figure 25: Actual vs Predicted output values for all calculated Global Sales prediction models

On Figure 25 above all plots are close to follow linear regression for the values closest to 0. However, the further we go for higher values, then more plots start to deviate from it (fastest deviating being LM an NNLS models). This proves that actually none of those models are good at predicting *Global_Sales* output.

3.3 Predicting North America sales

So if I can't obtain good model for predicting *Global_Sales* from *video_sales* data set, is there any other information that we can pull from it? Let's maybe focus on the most correlated with *Global_Sales* variable (see Figure 19), which was *NA_Sales*. I've excluded it from previous prediction models, as I didn't want to use any of regional sales for total sum prediction, but maybe following this path could lead to better results.

I can guess that trying to predict *NA_Sales* with the same set of parameters that I chose for *Global_Sales* would lead to similar results, so let's try different approach. What if we wanted to predict *NA_Sales* (so the biggest regional consumer - see Figure 4) for a game that was already released, gathered reviews from both Critics and Users AND we know how it performed in other regions? This way we would add more variables correlated strongly to the output we want to predict.

First, let's again create a new data frame for modeling, that consists of all our variables, without *Global_Sales*. This new data frame will be called *NAsales*.

```
NAsales <- video_sales %>% select(-Global_Sales)
```

Then, similarly to previous modeling for *Global_Sales*, I've created test and train sets, following Pareto Principle of 80/20 split.

```
na.test_index <- createDataPartition(y = NAsales$NA_Sales,
                                     times = 1, p = 0.2, list = FALSE)
na.test <- NAsales%>% slice(na.test_index)
na.train <- NAsales %>% slice(-na.test_index)
```

As a final preparation step before modeling, I've joined *na.train* and *na.test* sets on the non-numerical variables, to make sure that the same unique variables can be found in both sets, which will prevent models crashing due to near-zero variables importance in those variables. Using semi-join, we will be able to eliminate most of those instances (however *Rating* might still not have enough representatives in *EC* group).

```
na.train <- na.train %>%
  semi_join(na.test, by="Rating") %>%
  semi_join(na.test, by="Platform") %>%
  semi_join(na.test, by="Genre")
```

For comparison sake, I've used same regression models as during modeling *Global_Sales* output: Linear Regression, Cubist, Regression Trees, Random Forest and Non-Negative Least Squares. All models for *NA_Sales* prediction were also 10-fold cross-validated and pre-processed (scaled and centered variables).

First, let's model Linear Regression model and create prediction vector.

```
na.lm.fit <- train(NA_Sales~.,data=na.train, method="lm",
                   trControl=trainControl(method="cv",number=10),
                   preProcess = c("center", "scale"))

na.lm.pred <- predict(na.lm.fit,na.test)
```

Following that is the Cubist model. It has a possibility to tune *committees* and *neighbors* parameters, for which I set a range same as for previous Cubist modeling (integer from 1 to 9).

```
na.cubist.fit <- train(NA_Sales~.,data=na.train, method="cubist",
                       trControl=trainControl(method="cv",number=10),
                       tuneGrid=data.frame(committees=seq(1,9,1),
                                           neighbors=seq(1,9,1)),
                       preProcess = c("center", "scale"))

na.cubist.pred <- predict(na.cubist.fit,na.test)
```

Tuned parameters can be seen on the plot below, with highlighted best value for them that is used in final model. The best value of tuned parameter is then 5.

```
na.cubist.fit$bestTune
plot(na.cubist.fit,highlight = TRUE)
```

```
##   committees neighbors
## 5          5         5
```

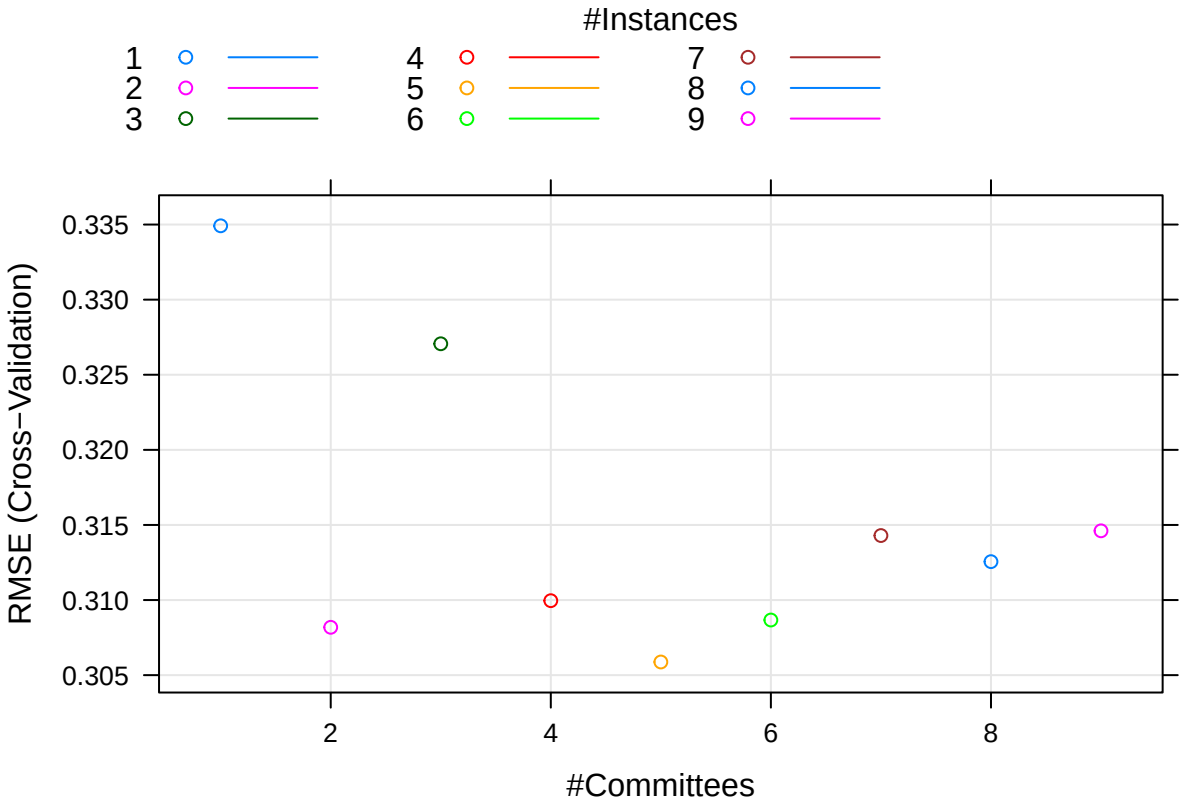

Figure 26: RMSE values for different tuned parameters for Cubist model of NA Sales

Next one is Regression Trees (rpart) model, where tunable parameter is *complexity parameter (cp)*, for which we also set same range of tuning as in predicting *Global_Sales* output.

```
na.rpart.fit <- train(NA_Sales~.,data=na.train, method="rpart",
                      tuneGrid=data.frame(cp=seq(0,0.05,0.005)),
                      trControl=trainControl(method="cv",number=10),
                      preProcess = c("center", "scale"))

na.rpart.pred <- predict(na.rpart.fit,na.test)
```

Similarly like in the *Global_Sales* predicting, cp parameter is equal to zero, which implies that the tree should not be pruned and be as complex as possible to achieve lowest RMSE.

This can be seen on tuning plot below, where *cp* parameter with lowest RMSE is highlighted.

```
na.rpart.fit$bestTune
```

```
##   cp
## 1  0
```

```
ggplot(na.rpart.fit,highlight = TRUE)
```

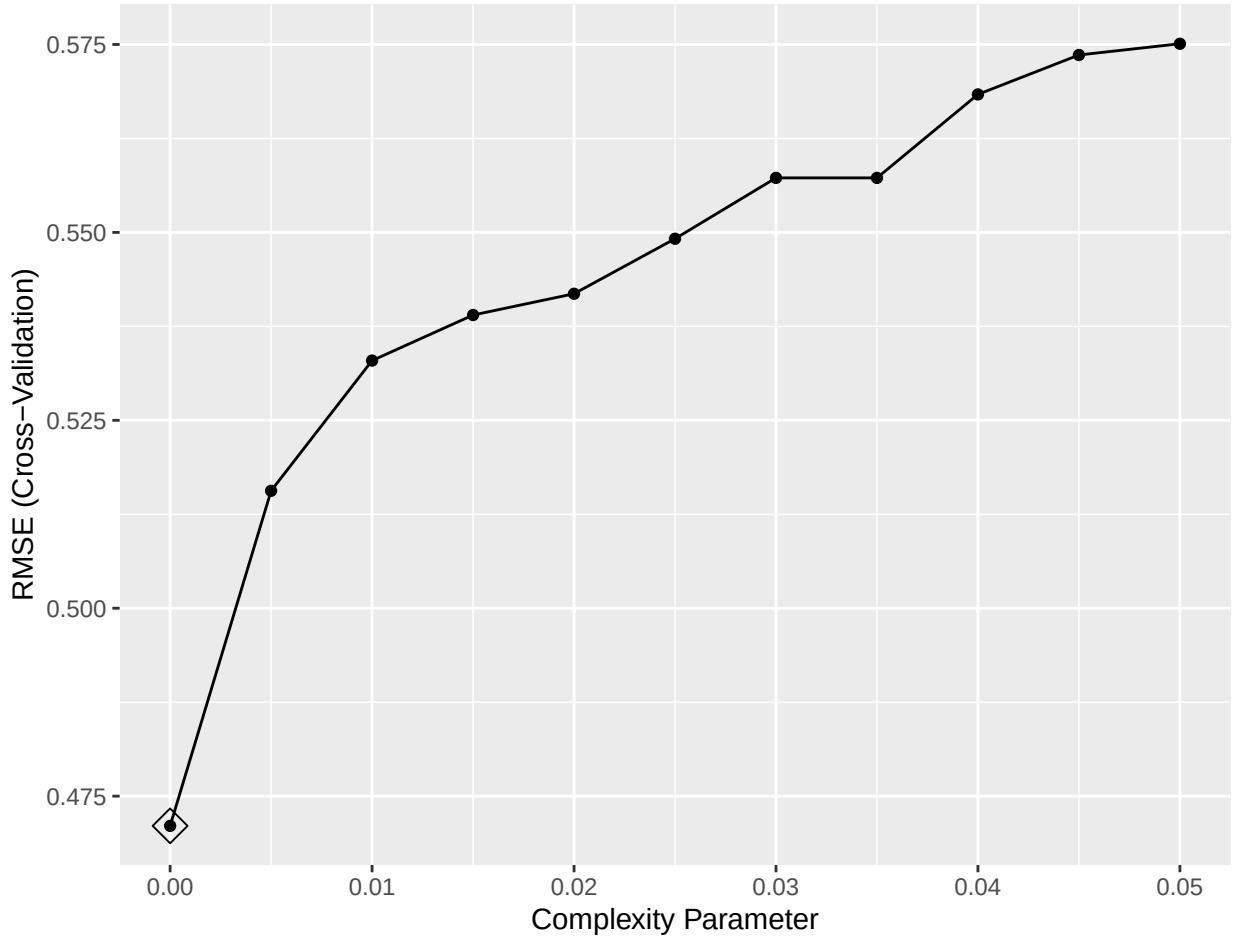

Figure 27: RMSE values for different tuned parameters for Rpart model of NA Sales

Fourth model that I will compare is Random Forest (rf) model. Besides tunable parameter *mtry* it can have modified number of trees made *ntree*. Similarly to before, for both of these I've set range and value same as for previous Random Forest modeling, equal to 100 trees and integer range from 1 to 25.

```
na.rf.fit <- train(NA_Sales~.,data=na.train, method="rf",
                   tuneGrid=data.frame(mtry=seq(1,25,1)), ntree=100,
                   trControl=trainControl(method="cv",number=10),
                   preProcess = c("center", "scale"))

na.rf.pred <- predict(na.rf.fit,na.test)
```

The best value of *mtry* that gives lowest RMSE is 19, which can be seen on plot below.

```
ggplot(na.rf.fit,highlight = TRUE)
```

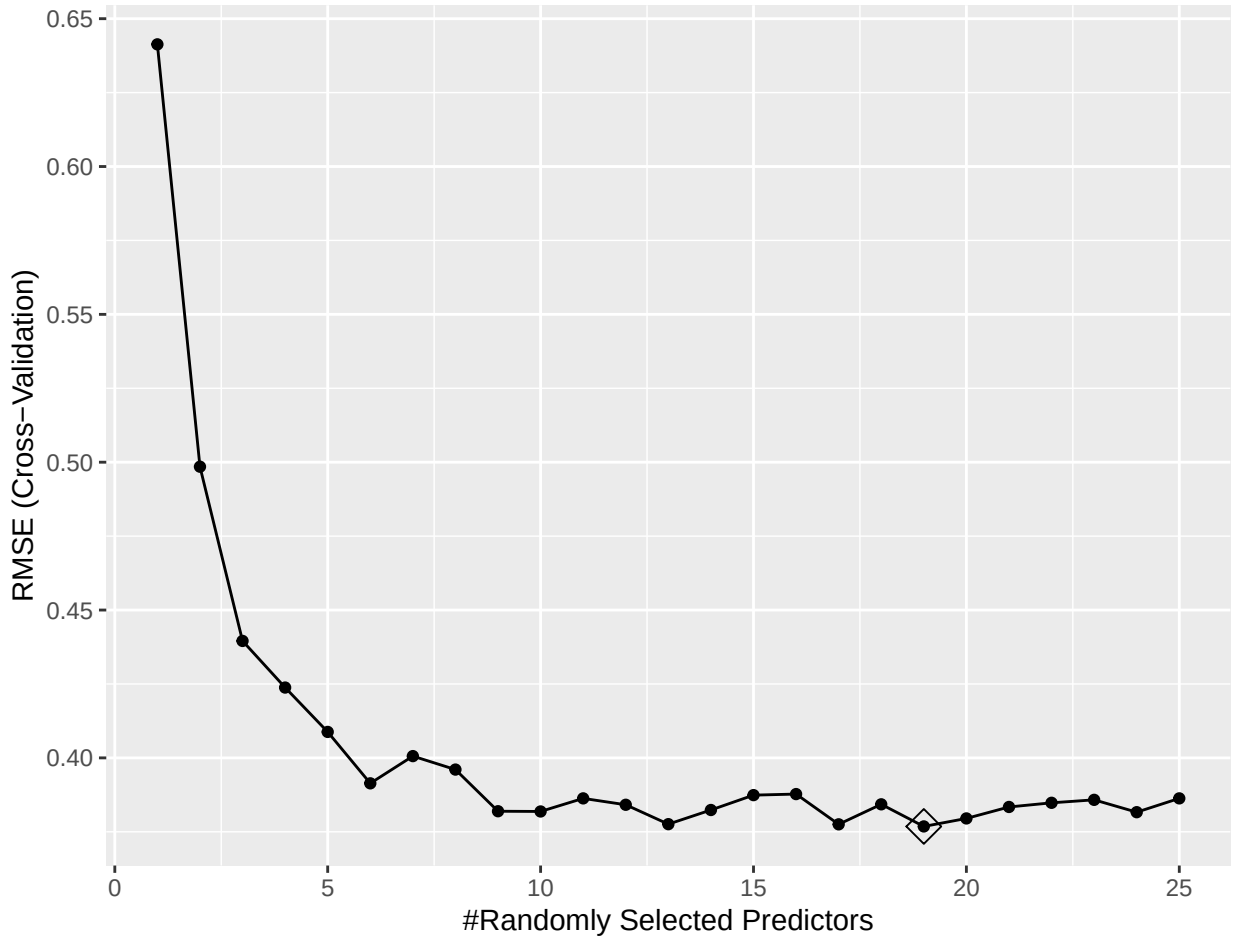

Figure 28: RMSE values for different tuned parameters for RF model of NA Sales

```
na.rf.fit$bestTune
```

```
##    mtry
## 19   19
```

The last model that I will be calculating is NNLS. For this one there were no tunable parameters.

```
na.nnls.fit <- train(NA_Sales~.,data=na.train, method="nnls",
                     trControl=trainControl(method="cv",number=10),
                     preProcess = c("center", "scale"))

na.nnls.pred <- predict(na.nnls.fit,na.test)
```

Now that all models for *NA_Sales* predictions have been computed, I've created a box plot containing RMSE, MAE and R-squared results for all of them.

```
models <- list(lm=na.lm.fit, cubist=na.cubist.fit, rpart=na.rpart.fit,
               randomforest=na.rf.fit, nnls=na.nnls.fit)

bwplot(resamples(models),metric=c("RMSE","Rsquared","MAE"),
       main="NA Sales learning models comparison")
```

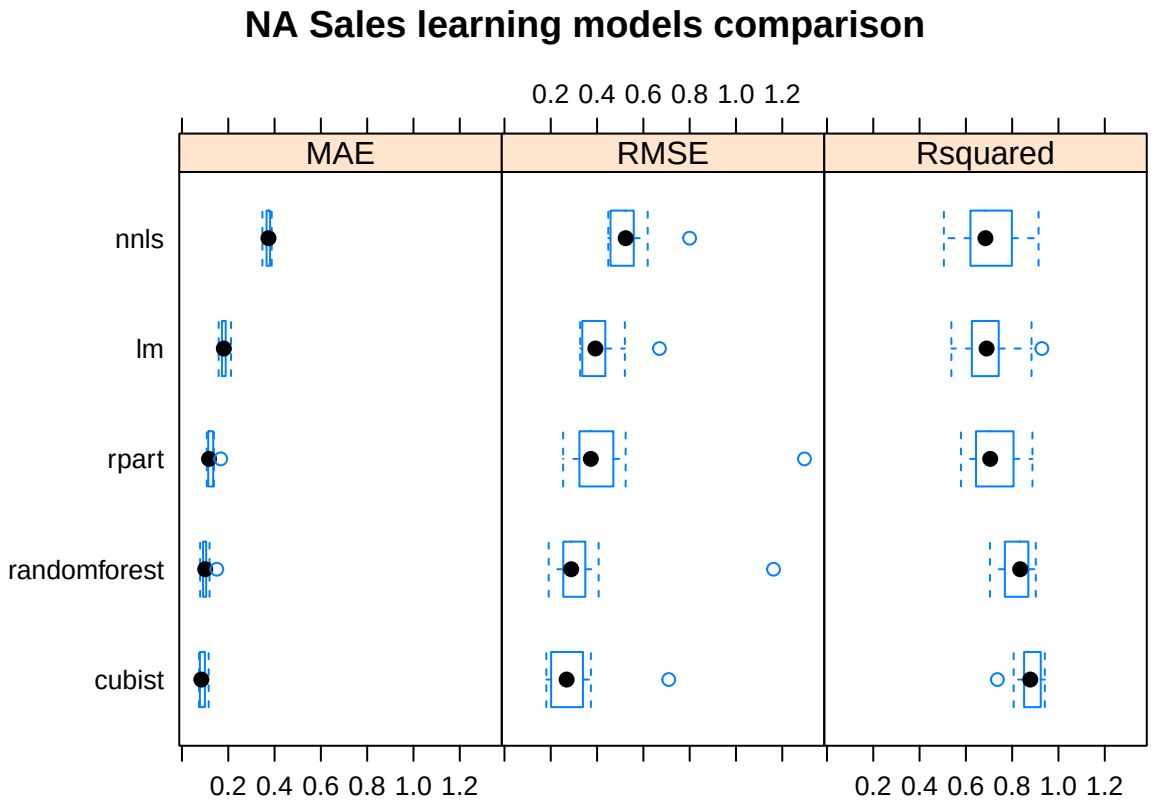

Figure 29: NA Sales learning models comparison

Above box plot clearly shows that all models have much better results than those for *Global_Sales* prediction. It also indicates that all of them would be statistically significant models (with R>0.65), but for the actual confirmation of that let's check more precise numbers.

Below I've computed median numbers of all three results indicators for all models computations (similarly like it was done for *Global_Sales*).

```
mae <- list(median(na.lm.fit$resample$MAE),
            median(na.cubist.fit$resample$MAE),
            median(na.rpart.fit$resample$MAE),
            median(na.rf.fit$resample$MAE),
            median(na.nnls.fit$resample$MAE))
rmse <- list(median(na.lm.fit$resample$RMSE),
             median(na.cubist.fit$resample$RMSE),
             median(na.rpart.fit$resample$RMSE),
             median(na.rf.fit$resample$RMSE),
```

```
             median(na.nnls.fit$resample$RMSE))
rsqr <- list(median(na.lm.fit$resample$Rsquared),
             median(na.cubist.fit$resample$Rsquared),
             median(na.rpart.fit$resample$Rsquared),
             median(na.rf.fit$resample$Rsquared),
             median(na.nnls.fit$resample$Rsquared))

na.results <- data.frame(Model=c("LM","Cubist","Rpart","RandomForest","NNLS"),
                         Median_MAE=unlist(mae), Median_RMSE=unlist(rmse),
                         Median_Rsquared=unlist(rsqr))
```

Table 14: Median MAE, RMSE and R-squared results for NA Sales prediction models

Model	Median_MAE	Median_RMSE	Median_Rsquared
LM	0.1808671	0.3927466	0.6895311
Cubist	0.0837192	0.2683861	0.8783241
Rpart	0.1168749	0.3731262	0.7051354
RandomForest	0.1003220	0.2882170	0.8345584
NNLS	0.3736104	0.5235990	0.6845805

From the table above it's now clear that all models were achieving much better results than during predicting *Global_Sales* output. The worst performance comes from NNLS model, with highest RMSE (0.52) and lowest R-Squared (0.68). The best one would be then Cubist model, with lowest RMSE (0.27), highest R-Squared (0.88) and lowest MAE (0.08).

Let's now take a look at Actual vs Predicted *NA_Sales* output plots below.

```
plot.lm <- data.frame(Actuals=na.test$NA_Sales,
                      Predicted=na.lm.pred) %>% mutate(Type="LM")
plot.cubist <- data.frame(Actuals=na.test$NA_Sales,
                          Predicted=na.cubist.pred) %>% mutate(Type="Cubist")
plot.rpart <- data.frame(Actuals=na.test$NA_Sales,
                         Predicted=na.rpart.pred) %>% mutate(Type="Rpart")
plot.rf <- data.frame(Actuals=na.test$NA_Sales,
                      Predicted=na.rf.pred) %>% mutate(Type="RandomForest")
plot.nnls <- data.frame(Actuals=na.test$NA_Sales,
                        Predicted=na.nnls.pred) %>% mutate(Type="NNLS")

final.plot <- rbind(plot.lm,plot.cubist,plot.rpart,plot.rf,plot.nnls)
```

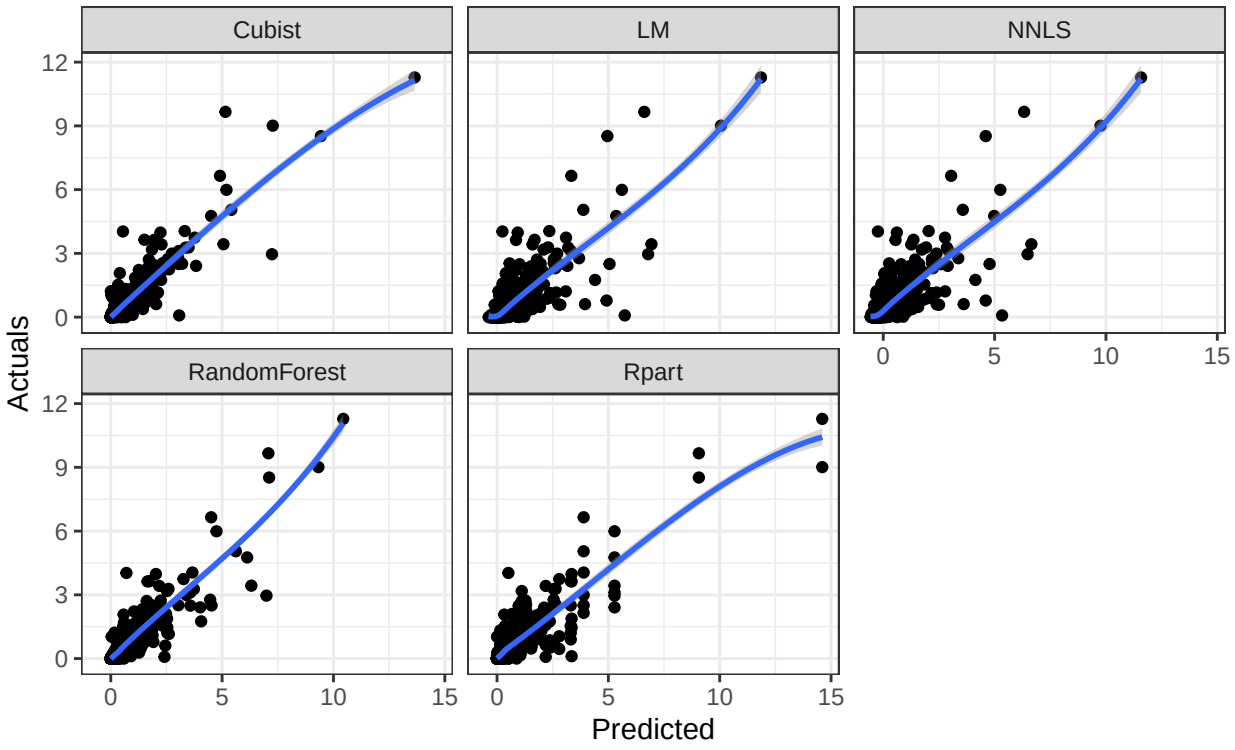

Figure 30: Actual vs Predicted output values for all calculated Global Sales prediction models

Almost all plots above show much straighter smoothed lines, almost linear in lower ranges. The best ones are also Cubist and Random Forest, while the worst ones (having slight curve even in the lowest range) are NNLS and LM. Both of those cases prove what was shown already in Table 14, where Cubist and Random Forest are the best performing, while NNLS and LM are the worst.

Knowing now that all chosen models could be used (with better or worse results) for predicting *NA_Sales*, let's take final look into variable importance for each of them. On Table 15 below I've created a summary of top 10 most important variables for each model, using *varImp* function.

```
na.lm.imp <- varImp(na.lm.fit)
na.cubist.imp <- varImp(na.cubist.fit)
na.rpart.imp <- varImp(na.rpart.fit)
na.rf.imp <- varImp(na.rf.fit)
na.nnls.imp <- varImp(na.nnls.fit)
```

Table 15: Variables importance in five checked regression models for NA Sales

Best performing models				Other models					
Cubist		RF		LM		Rpart		NNLS	
Variable	Overall	Variable	Overall	Variable	Overall	Variable	Overall	Variable	Overall
Other_Sales	100.00000	EU_Sales	100.000000	EU_Sales	100.000000	EU_Sales	100.00000	EU_Sales	100.00000
EU_Sales	91.42857	Other_Sales	97.933357	Other_Sales	39.090762	Developer.fct	97.25082	Other_Sales	35.56223
Year_of_Release	68.00000	Developer.fct	19.737967	Developer.fct	16.502285	Year_of_Release	73.27221	PlatformX360	20.91651
PlatformPS2	48.57143	JP_Sales	15.614510	Year_of_Release	9.066643	Publisher.fct	71.96284	PlatformWii	17.73288
Developer.fct	48.00000	Critic_Score	9.499359	PlatformPS4	7.028876	Critic_Count	69.49802	PlatformDS	15.00901
PlatformPC	30.85714	Critic_Count	5.864851	PlatformX360	6.898848	Critic_Score	61.02782	Developer.fct	14.19117
PlatformPS3	23.42857	User_Count	5.362732	Publisher.fct	5.598833	User_Count	58.33182	PlatformXB	13.21500
PlatformPSP	18.85714	Publisher.fct	4.258035	PlatformPC	4.796994	User_Score	47.60781	PlatformPS2	12.84332
Critic_Score	17.71429	Year_of_Release	3.562496	PlatformWii	4.756502	Other_Sales	32.19622	PlatformGC	11.29495
PlatformX360	17.71429	PlatformX360	3.360399	GenreFighting	4.689779	JP_Sales	22.15382	PlatformGBA	10.26151

What is clearly visible from above Table is that for the best models (which are Cubist and RF) most of variance is almost equally explained by both *Other_Sales* and *EU_Sales* variables (both oscillating near 90-100%). For the rest of them only one variable is outstanding with it's importance - *EU_Sales* explaining 100% variance in all three models of LM, Rpart and NNLS.

None other variable and its levels are consistent in their importance in all five considered models, and some (like *Genre* and *Rating*) are actually not in top 10 at all.

4 Summary & conclusions

When talking about *video_sales* data set and prediction models calculated from it, we need to be remember that if we were trying to predict games sales in reality, more research into unknown variables would have to be conducted. Those missing variables are currently creating "noise" in our models, and including them could lead to improvement in prediction accuracy. However, most of them would be either hard to get or requiring joining few other new sources to existing *video_sales* data set. Looking into real world data, the most obvious missing variables that could be big influencers are:

- cost spend on marketing (was it global campaign, were any known faces involved)
- socio-political changes of the region (is there an economical depression or pandemic happening)
- social media influencers generating organic hype (was this game featured in un-sponsored "let's play" by known streamer or Youtuber)
- demographics of players (age, gender, nationality, ethnicity)
- price of the game and/or platform needed to play

Another thing that needs to be noted is that I've assumed in prediction models that we're trying to find out final sales value *after* game has been released. Due to that assumptions User and Critic variables (*User_Score*, *User_Count*, *Critic_Score* and *Critic_Count*) have been included in the models. If models were to be used in prediction of sales upon game release day, those variables would have to be excluded, as they are not known at that point of time (especially User ones, as there might be few Critics with pre-release access). In this case all models for both *Global_Sales* and *NA_Sales* would have to be recalculated.

Regarding machine learning modeling done for two different outputs, it is clear that *NA_Sales* prediction models have been performing much better than those of *Global_Sales*. *NA_Sales* has main difference of including also other regional sales values, being strongly correlated with desired output. All 5 models for this configuration have performed well and could be used to predict actual *NA_Sales* values. However, *Global_Sales* models were not critically bad, and I strongly believe it's possible to use at least Cubist and Random Forest models for this prediction, with disclaimer for double-checking actual results. The performance of the models in both cases could be improved if we also take into consideration mentioned before different configuration of input variables, include models that put higher importance on the unexplained noise or enlarger the data set with more up-to-date values.

The *Complexity Parameter (cp)* in Regression Tree models (Rpart) in both instances had a best fit of 0. The general rule is that the smaller *cp* value the better. This can indicate that I've been really successful in preparing the *video_sales* data set before and it doesn't require pruning after that, but it may also indicate that we're close to having models being overfitted as the data set is too small. Seeing as in both prediction cases *rpart* model was not the best performing one when confronted with test set, that could indicate that either this is really not the best model for this kind of data or we do really have a case of overfitting that is not so visible in other models.

On the other hand, we also have an example of *Rating* variable having near-zero variance (for *EC* level) for both prediction cases, despite not showing such tendencies before modeling. Not a single level of this variable is in top 10 most important variables (see Table 15). This could be an indication that this *Rating* level has too little instances and therefore the data set on which we are training the models is too small. As mentioned before, this could be also avoided by removing *Rating* from the variable used for predicting, with the cost of slightly worse results (which are already quite satisfying).

In summary, possible **next steps for video game sales prediction models improvement** could be:

- considering adding data after 2016 to the data set to make it larger,
- analyze actual gaming related data and events happening through time to understand undiscovered influencers, and possibly include them in prediction models,
- removing *Rating* variable as predictor, since it has very low (and almost zero in some instances) variance and importance,
- to predict sales expected at the day of release of the game remove User (and possibly also Critic) Score and Count variables.

5 Appendix

5.1 Session info

```
sessionInfo()
```

```
## R version 4.0.1 (2020-06-06)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18363)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=Polish_Poland.1250  LC_CTYPE=Polish_Poland.1250
## [3] LC_MONETARY=Polish_Poland.1250 LC_NUMERIC=C
## [5] LC_TIME=Polish_Poland.1250
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] nnls_1.4           Cubist_0.2.3       gridExtra_2.3      wordcloud_2.6
##  [5] RColorBrewer_1.1-2 tm_0.7-7           NLP_0.2-1          ggpubr_0.4.0
##  [9] scales_1.1.1       corrgram_1.13      caret_6.0-86       funModeling_1.9.4
## [13] Hmisc_4.4-1        Formula_1.2-4      survival_3.1-12    lattice_0.20-41
## [17] forcats_0.5.0      stringr_1.4.0      dplyr_1.0.2        purrr_0.3.4
## [21] readr_1.4.0        tidyr_1.1.2        tibble_3.0.4       ggplot2_3.3.2
## [25] tidyverse_1.3.0    kableExtra_1.3.1   knitr_1.30
##
## loaded via a namespace (and not attached):
##   [1] colorspace_1.4-1     ggsignif_0.6.0       rio_0.5.16
##   [4] ellipsis_0.3.1       class_7.3-17         htmlTable_2.1.0
##   [7] base64enc_0.1-3      fs_1.5.0             rstudioapi_0.12
##  [10] farver_2.0.3         prodlim_2019.11.13   fansi_0.4.1
##  [13] lubridate_1.7.9      xml2_1.3.2           codetools_0.2-16
##  [16] splines_4.0.1        jsonlite_1.7.1       entropy_1.2.1
##  [19] pROC_1.16.2          broom_0.7.2          cluster_2.1.0
##  [22] dbplyr_2.0.0         png_0.1-7            compiler_4.0.1
##  [25] httr_1.4.2           backports_1.2.0      assertthat_0.2.1
##  [28] Matrix_1.2-18        lazyeval_0.2.2       cli_2.1.0
##  [31] htmltools_0.5.0      tools_4.0.1          gtable_0.3.0
##  [34] glue_1.4.2           reshape2_1.4.4       Rcpp_1.0.5
##  [37] slam_0.1-47          carData_3.0-4        cellranger_1.1.0
##  [40] vctrs_0.3.4          nlme_3.1-148         iterators_1.0.13
##  [43] timeDate_3043.102    xfun_0.19            gower_0.2.2
##  [46] openxlsx_4.2.3       rvest_0.3.6          lifecycle_0.2.0
##  [49] rstatix_0.6.0        MASS_7.3-51.6        ipred_0.9-9
##  [52] TSP_1.1-10           hms_0.5.3            parallel_4.0.1
##  [55] curl_4.3             yaml_2.2.1           pander_0.6.3
##  [58] rpart_4.1-15         latticeExtra_0.6-29  stringi_1.5.3
##  [61] highr_0.8            randomForest_4.6-14  foreach_1.5.1
##  [64] checkmate_2.0.0      seriation_1.2-9      zip_2.1.1
```

```
##  [67] lava_1.6.8.1         rlang_0.4.8          pkgconfig_2.0.3
##  [70] moments_0.14         evaluate_0.14        ROCR_1.0-11
##  [73] labeling_0.4.2       recipes_0.1.15       htmlwidgets_1.5.2
##  [76] cowplot_1.1.0        tidyselect_1.1.0     plyr_1.8.6
##  [79] magrittr_1.5         R6_2.5.0             generics_0.1.0
##  [82] DBI_1.1.0            mgcv_1.8-31          pillar_1.4.6
##  [85] haven_2.3.1          foreign_0.8-80       withr_2.3.0
##  [88] abind_1.4-5          nnet_7.3-14          car_3.0-10
##  [91] modelr_0.1.8         crayon_1.3.4         utf8_1.1.4
##  [94] rmarkdown_2.5        jpeg_0.1-8.1         grid_4.0.1
##  [97] readxl_1.3.1         data.table_1.13.2    ModelMetrics_1.2.2.2
## [100] reprex_0.3.0         digest_0.6.27        webshot_0.5.2
## [103] stats4_4.0.1         munsell_0.5.0        registry_0.5-1
## [106] viridisLite_0.3.0
```

5.2 Full code

```
# This code contains data exploration and prediction models on video game sales.
# Data set used here is Video Game Sales with Rating data set.
# This data set has been created by Rush Kirubi on Kaggle
# Created in Dec 2020.


# 1. Download and prepare the data set ----------------------------------

# Ensuring all packages are installed and loaded
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(funModeling)) install.packages("funModeling",
                                           repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(corrgram)) install.packages("corrgram",
                                        repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales",
                                      repos = "http://cran.us.r-project.org")
if(!require(ggpubr)) install.packages("ggpubr",
                                      repos = "http://cran.us.r-project.org")
if(!require(tm)) install.packages("tm",
                                  repos = "http://cran.us.r-project.org")
if(!require(wordcloud)) install.packages("wordcloud",
                                         repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra",
                                         repos = "http://cran.us.r-project.org")
# For model in train function
if(!require(Cubist)) install.packages("Cubist",
                                      repos = "http://cran.us.r-project.org")
if(!require(nnls)) install.packages("nnls",
                                    repos = "http://cran.us.r-project.org")

library(tidyverse)
library(funModeling)
```

```r
library(scales)
library(caret)
library(corrgram)
library(ggpubr)
library(wordcloud)
library(tm)
library(gridExtra)

# Downloading Video Game Sales with Rating data set,
# created by Rush Kirubi from my GitHub repository
url_path <-
  "https://raw.githubusercontent.com/Golfik/Video-Games-Sales/master/
Video_Games_Sales_as_at_22_Dec_2016.csv"

raw.videosales <- read.csv(url(url_path),na.strings = c("","NA"))
rm(url_path)

# Create a duplicate of data_set for analysis (leaving raw.videosales for reference)
video_sales <- raw.videosales



# 2. Data overview ----------------------------------------------------

# First look in the data set
glimpse(video_sales)

# Checking out duplicate values - non found
duplicated(video_sales) %>% sum()

# Looking deeper to see if there are any obvious outliers
summary(video_sales)

# Checking out if there are many NAs and how many unique values
status(video_sales)



# 3. Exploring, visualizing and cleaning the data set ----------------------


# 3.1. Global and regional sales variables ----

# Analyzing density of Global Sales
ggplot(video_sales,aes(Global_Sales)) +
  geom_histogram(aes(y=..density..),binwidth = 0.5) +
  geom_density(alpha=0.2,fill="#EF3B2C") +
  labs(x="Global Sales [M units]", y ="Density",
       title="Density plot of Global Sales variable") +
  theme_bw()

# Checking out top 10 best selling games globally,
# to see what outliers may influence long tail on previous plot
```

```r
video_sales %>%
  select(Name,Platform, Genre, User_Count, Global_Sales) %>%
  arrange(desc(Global_Sales)) %>%
  head(10)

# Global Sales density plot of all up to 3rd quantile,
# with shown mean value (zooming to values without tail)
tot.mean <- video_sales %>%
  arrange(desc(Global_Sales)) %>%
  select(Name, Global_Sales) %>%
  summarise(tot.mean=mean(Global_Sales)) %>%
  unlist()
video_sales %>%
  arrange(desc(Global_Sales)) %>%
  filter(between(Global_Sales,0,0.47)) %>%
  ggplot(aes(Global_Sales)) +
  geom_histogram(aes(y=..density..),binwidth = 0.01) +
  geom_density(alpha=0.2,fill="#EF3B2C") +
  geom_vline(aes(xintercept=tot.mean), linetype="dashed", size=1) +
  labs(x="Global Sales [M units]", y ="Density",
       title="Density of Global Sales, exluding top quantile",
       subtitle="Dashed line marking mean value of 0.5335 (all values)") +
  theme_bw()
rm(tot.mean)

# Changing the Year of Release to numeric type
video_sales <- video_sales %>%
  mutate(Year_of_Release=as.numeric(Year_of_Release))

# Global Sales, profit per title and number of games through years
plot1 <- video_sales %>%
  group_by(Year_of_Release) %>%
  summarise(sum_of_sales=sum(Global_Sales)) %>%
  ggplot() +
  geom_point(aes(x=Year_of_Release, y=sum_of_sales),
             stat = 'identity',show.legend = FALSE) +
  geom_line(aes(x=Year_of_Release, y=sum_of_sales),
            stat = 'identity',show.legend = FALSE) +
  labs(x = "Year", y = "Global Sales", title="Global sales [M units] through years") +
  theme_bw()
plot2 <- video_sales %>%
  group_by(Year_of_Release) %>%
  summarise(sales_pertitle=sum(Global_Sales)/n()) %>%
  ggplot() +
  geom_point(aes(x=Year_of_Release, y=sales_pertitle),
             stat = 'identity',show.legend = FALSE) +
  geom_line(aes(x=Year_of_Release, y=sales_pertitle),
            stat = 'identity',show.legend = FALSE) +
  labs(x = "Year", y = "Sales per title",
       title="Units sold [M units] per game title through years") +
  theme_bw() +
  scale_color_brewer(palette = "RdYlBu")
plot3 <- video_sales %>%
```

```
  group_by(Year_of_Release) %>%
  summarise(n=n()) %>%
  ggplot() +
  geom_point(aes(x=Year_of_Release, y=n),
             stat = 'identity',show.legend = FALSE) +
  geom_line(aes(x=Year_of_Release, y=n),
            stat = 'identity',show.legend = FALSE) +
  labs(x = "Year", y = "Number of games",
       title="Number of games released through years") +
  theme_bw()
ggarrange(plot1,plot2,plot3,ncol=1, align="hv")
rm(plot1,plot2,plot3)

# Remove 2020 as weird and 2017 as not enough data (insight from previous graph),
# as well as the name of original data set suggest is shouldn't be included
video_sales %>%
  group_by(Year_of_Release) %>%
  summarise(n=n()) %>%
  arrange(desc(Year_of_Release))
video_sales <- video_sales %>% filter(Year_of_Release < 2017)

# Share of sales between regions through years
video_sales %>%
  select(Year_of_Release, NA_Sales, EU_Sales, JP_Sales, Other_Sales) %>%
  gather(Region, Sales, NA_Sales, EU_Sales, JP_Sales, Other_Sales) %>%
  group_by(Year_of_Release,Region) %>%
  ggplot() +
  geom_bar(aes(x=Year_of_Release, y=Sales, fill=Region),
           position="fill", stat="identity") +
  labs(x = "Year", y = "Sales",
       title="Share of sales between regions through years") +
  theme_bw() +
  scale_y_continuous(labels = percent_format()) +
  scale_fill_brewer(palette = "RdYlBu",
                    labels=c("EU", "Japan", "North America", "Other"))


# 3.2. Publisher and Developer analysis ------------------------------------

# Global sales by publisher, as total sum of sales made
plot1 <- video_sales %>%
  gather(Region, Sales, NA_Sales, EU_Sales, JP_Sales, Other_Sales) %>%
  group_by(Publisher,Region) %>%
  summarise(sum_Sales=sum(Sales),total=sum(Global_Sales)) %>%
  arrange(desc(total)) %>%
  head(40) %>%
  ggplot(aes(x=reorder(Publisher,desc(total)), y=sum_Sales,fill=Region)) +
  geom_bar(stat = "identity") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 15, hjust=0.95)) +
  labs(x="Publisher", y ="Global Sales [M units]",
       title="Top 10 best selling publishers globally, as total sales made") +
  scale_fill_brewer(palette = "PRGn", labels=c("EU", "Japan", "North America", "Other"))
```

```r
# Global Sales by developer, as total sum of sales
plot2 <- video_sales %>%
  filter(!is.na(Developer)) %>%
  gather(Region, Sales, NA_Sales, EU_Sales, JP_Sales, Other_Sales) %>%
  group_by(Developer,Region) %>%
  summarise(sum_Sales=sum(Sales),total=sum(Global_Sales)) %>%
  arrange(desc(total)) %>%
  head(40) %>%
  ggplot(aes(x=reorder(Developer,desc(total)), y=sum_Sales,fill=Region)) +
  geom_bar(stat = "identity") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 15, hjust=0.95)) +
  labs(x="Developer", y ="Global Sales [M units]",
       title="Top 10 best selling developers globally, as total sales made") +
  scale_fill_brewer(palette = "PRGn", labels=c("EU", "Japan", "North America", "Other"))

# Arranging two previous plots, and clearing environment
ggarrange(plot1,plot2,ncol=1,nrow=2,common.legend=TRUE, legend="bottom")
rm(plot,plot2)

# Global sales made per title by top publishers
plot1 <- video_sales %>%
  group_by(Publisher) %>%
  summarise(sale_pertitle=sum(Global_Sales)/n()) %>%
  arrange(desc(sale_pertitle)) %>%
  head(10) %>%
  ggplot(aes(y=sale_pertitle,x=reorder(Publisher,sale_pertitle))) +
  geom_bar(stat = "identity", fill="#006CD1") +
  theme_bw() +
  labs(x="Publisher", y ="Sales per title [M units]",
       title="Top 10 publishers by sales per title") +
  coord_flip() +
  scale_x_discrete(labels = function(x) str_wrap(x, width = 50))

# Global sales per title by top developers
plot2 <- video_sales %>%
  filter(!is.na(Developer)) %>%
  group_by(Developer) %>%
  summarise(sale_pertitle=sum(Global_Sales)/n()) %>%
  arrange(desc(sale_pertitle)) %>%
  head(10) %>%
  ggplot(aes(y=sale_pertitle,x=reorder(Developer,sale_pertitle))) +
  geom_bar(stat = "identity", fill="#006CD1") +
  theme_bw() +
  labs(x="Developer", y ="Sales per title [M units]",
       title="Top 10 developers by sales per title") +
  coord_flip() +
  scale_x_discrete(labels = function(x) str_wrap(x, width = 50))

# Arranging two previous plots, and clearing environment
ggarrange(plot1,plot2,ncol=1,nrow=2, align="v")
rm(plot1,plot2)
```

```
# How many games has top Publisher released?
video_sales %>%
  group_by(Publisher) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n()) %>%
  arrange(desc(sale_per_title)) %>%
  select(Publisher,sale_per_title, no_of_games) %>%
  unique() %>%
  head(10)

# How many games has top Developer released?
video_sales %>%
  group_by(Developer) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n()) %>%
  arrange(desc(sale_per_title)) %>%
  select(Developer,sale_per_title, no_of_games) %>%
  unique() %>%
  head(10)

# How will publisher factor look like if we penalize studios with small number
# of titles with function with limit to 1?
video_sales %>%
  group_by(Publisher) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n(),
         adj.factor=no_of_games*sin(1/no_of_games),
         Publisher.fct=sale_per_title*no_of_games*sin(1/no_of_games)) %>%
  arrange(desc(sale_per_title)) %>%
  select(Publisher,sale_per_title, no_of_games, adj.factor, Publisher.fct) %>%
  unique() %>%
  head(10)

# How will developer factor look like if we penalize studios with small number
# of titles with function with limit to 1?
video_sales %>%
  group_by(Developer) %>%
  mutate(sale_per_title=sum(Global_Sales)/n(),no_of_games=n(),
         adj.factor=no_of_games*sin(1/no_of_games),
         Developer.fct=sale_per_title*no_of_games*sin(1/no_of_games)) %>%
  arrange(desc(sale_per_title)) %>%
  select(Developer,sale_per_title, no_of_games, adj.factor, Developer.fct) %>%
  unique() %>%
  head(10)

# Add new Developer and Publisher factors to the video_sales data set,
# and removing original Publisher and Developer
video_sales <- video_sales %>%
  group_by(Publisher) %>%
  mutate(Publisher.fct=(sum(Global_Sales)/n())*n()*sin(1/n())) %>%
  ungroup()
video_sales <- video_sales %>%
  group_by(Developer) %>%
  mutate(Developer.fct=(sum(Global_Sales)/n())*n()*sin(1/n())) %>%
  ungroup()
video_sales <- video_sales %>%
```

```
  select(-Publisher,-Developer)


# 3.3. Platform and Genre analysis ----------------------------------------

# Number of game titles released per platform (only top)
video_sales %>%
  group_by(Platform) %>%
  summarise(n=n()) %>%
  arrange(desc(n)) %>%
  head(15) %>%
  ggplot(aes(x=reorder(Platform,n),y=n)) +
  geom_segment(aes(x=reorder(Platform,n),
                   xend=reorder(Platform,n), y=0, yend=n), color="grey") +
  geom_point(color="#FD8D3C", size=4) +
  coord_flip() +
  theme_bw() +
  labs(x="Platform", y ="Number of titles released",
       title="Top 15 platforms with highest number of games released")

# Top selling games, and their genres and platforms
video_sales %>%
  arrange(desc(Global_Sales)) %>%
  head(15) %>%
  ggplot(aes(y=Global_Sales,x=reorder(Name,Global_Sales),fill=Genre)) +
  geom_bar(stat="identity") + geom_text(aes(label=Platform), hjust=1.5) +
  coord_flip() +
  theme_bw() +
  labs(x="Global Sales [M of units]", y ="Game title and its platform",
       title="Top 15 selling games globally") +
  scale_fill_brewer(palette = "RdYlBu")

# Sales per genre
video_sales %>%
  group_by(Genre) %>%
  summarise(n_titles=n(), total_sales=sum(Global_Sales)) %>%
  arrange(desc(total_sales))

# Erasing NAs from Genre
video_sales <- video_sales %>%
  filter(!is.na(Genre))

# Changing Genre to a factor
video_sales$Genre <- as.factor(video_sales$Genre)

# Average sales per one title by each game genre
video_sales %>%
  group_by(Genre) %>%
  summarise(sale_pertitle=sum(Global_Sales)/n()) %>%
  arrange(desc(sale_pertitle)) %>%
  ggplot(aes(y=sale_pertitle,x=reorder(Genre,desc(sale_pertitle)))) +
  geom_bar(stat = "identity", fill="#006CD1") +
  theme_bw() +
```

```
  labs(x="Genre", y ="Sales per title [M units]",
       title="Sales per one title per game genre") +
  theme(axis.text.x = element_text(angle = 15, hjust=0.95))


# 3.4. User and Critic Score and Count analysis --------------------------

# User Score and Critic Score frequency plot,
# across different Genre (dark theme for visibility)
plot1 <- ggplot(video_sales, aes(as.numeric(User_Score), color=Genre)) +
  geom_freqpoly(size=1) +
  theme_dark() +
  scale_color_brewer(palette = "Paired") +
  labs(x="User Score", y ="Count",
       title="Distribution of User Scores across game genres")
plot2 <- ggplot(video_sales, aes(as.numeric(Critic_Score), color=Genre)) +
  geom_freqpoly(size=1) +
  theme_dark() +
  scale_color_brewer(palette = "Paired") +
  labs(x="Critic Score", y ="Count",
       title="Distribution of Critic Scores across game genres")
ggarrange(plot1,plot2,ncol=1,nrow=2,common.legend=TRUE, legend="bottom")
rm(plot1,plot2)

# Checking out User_Score - why it's character,
# then change tbd to NA, and variable type to numeric
video_sales %>%
  group_by(User_Score) %>%
  summarise(n=n()) %>%
  arrange(desc(n)) %>%
  head(10)

video_sales$User_Score[video_sales$User_Score=="tbd"] <- NA

video_sales$User_Score <- as.numeric(video_sales$User_Score)

#Check the same for Critic Score.
# There are no character values, and it can be changed to numeric
video_sales %>%
  group_by(Critic_Score) %>%
  summarise(n=n()) %>%
  arrange(desc(n)) %>%
  head(10)

video_sales$Critic_Score <- as.numeric(video_sales$Critic_Score)

# Correlation plot between User Score and Critic Score
video_sales %>%
  na.omit() %>%
  ggplot(aes(x=User_Score,y=Critic_Score)) +
  geom_point() +
  geom_smooth(method="loess") +
  theme_bw() +
```

```
  labs(x="User Score", y ="Critic Score",
       title="Correlation between User and Critic Score")

# Influence of scoring difference between User and Critic Score on Global Sales
video_sales %>%
  na.omit() %>%
  mutate(diff=(User_Score*10-Critic_Score)) %>%
  group_by(diff) %>%
  summarise(sums=sum(Global_Sales)) %>%
  ggplot(aes(x=diff,y=sums)) +
  geom_point(color="#006CD1") +
  theme_bw() +
  labs(x="Difference between User and Critic Score",
       y ="Global Sales [M of units]",
       title="Influence of scoring difference between Users and Critics on Global Sales")

# User and Critic Score impact on sales
video_sales %>%
  na.omit() %>%
  mutate(User_Score=User_Score*10) %>%
  gather(Type, Score, User_Score, Critic_Score) %>%
  group_by(Type,Score) %>%
  summarise(totalsales=sum(Global_Sales)) %>%
  ggplot(aes(x=Score,y=totalsales, fill=Type)) +
  geom_area(position="identity") +
  facet_grid(Type~.) +
  theme_bw() +
  scale_fill_brewer(palette="Dark2", label=c("Critic Score", "User Score*10")) +
  labs(x="Score", y ="Global Sales [M of units]",
       title="Global Sales vs. Critic and User Score")

# Count analysis of users and critics, and their influence on Global Sales
video_sales %>%
  na.omit() %>%
  gather(Type, Count, User_Count, Critic_Count) %>%
  group_by(Type,Count) %>%
  summarise(totalsales=sum(Global_Sales)) %>%
  ggplot(aes(x=Count,y=totalsales,fill=Type)) +
  geom_area(position="identity") +
  facet_grid(~Type, scales="free") +
  theme_bw() +
  labs(x="Count", y ="Global Sales [M of units]",
       title="Global Sales vs. Critic and User score Count") +
  scale_fill_brewer(palette="Dark2", label=c("Critic Count", "User Count"))

# User and Critic Score (as average) and Count across years
plot1 <- video_sales %>%
  na.omit() %>%
  group_by(Year_of_Release) %>%
  summarise(totalusers=mean(User_Score*10/User_Count),
            totalcritics=mean(Critic_Score/Critic_Count)) %>%
  gather(Type, Count, totalusers, totalcritics) %>%
  ggplot() +
```

```
  geom_line(aes(x=Year_of_Release, y=Count, color=Type), size=1) +
  theme_bw() +
  scale_color_brewer(palette = "Dark2", label=c("Critics", "Users")) +
  labs(x="Year of Release", y ="Sum of avg Scores (normalized) ",
       title="Distribution of Critic and User Scores (avg per title) across years")
plot2 <- video_sales %>%
  na.omit() %>%
  group_by(Year_of_Release) %>%
  summarise(totalusers=sum(User_Count),
            totalcritics=sum(Critic_Count)) %>%
  gather(Type, Count, totalusers, totalcritics) %>%
  ggplot() +
  geom_line(aes(x=Year_of_Release, y=Count, color=Type), size=1) +
  theme_bw() +
  scale_color_brewer(palette = "Dark2", label=c("Critics", "Users")) +
  labs(x="Year of Release", y ="Sum of Counts",
       title="Distribution of Critic and User score Counts across years")
ggarrange(plot1,plot2,ncol=1,nrow=2,common.legend=TRUE, legend="bottom", align="v")
rm(plot1,plot2)

# Are there any titles that has not been scored by Users or by Critics?
video_sales %>% filter(User_Count==0 | Critic_Count==0) %>%
  select(Name,Year_of_Release,Global_Sales,User_Score,User_Count,
         Critic_Score,Critic_Count)

# Imputing User and Critic Score and Count NAs with medians
video_sales$User_Score[is.na(video_sales$User_Score)] <-
  median(video_sales$User_Score, na.rm = TRUE)
video_sales$User_Count[is.na(video_sales$User_Count)] <-
  median(video_sales$User_Count, na.rm = TRUE)
video_sales$Critic_Score[is.na(video_sales$Critic_Score)] <-
  median(video_sales$Critic_Score, na.rm = TRUE)
video_sales$Critic_Count[is.na(video_sales$Critic_Count)] <-
  median(video_sales$Critic_Count, na.rm = TRUE)


# 3.5. Rating variable ----------------------------------------------------

# Exploring unique values of Rating variable
video_sales %>% group_by(Rating) %>% summarise(n=n()) %>% arrange(desc(n))

# Changing the K-A rating to E, and RP to NA value
video_sales$Rating[video_sales$Rating=="RP"] <- NA
video_sales$Rating[video_sales$Rating=="K-A"] <- "E"

# Chaining Rating variable to factor
video_sales$Rating <- as.factor(video_sales$Rating)

# Analysis of game rating influence on global sales per one title
video_sales %>%
  filter(!is.na(Rating)) %>%
  group_by(Rating) %>%
  summarise(sales=sum(Global_Sales)/n(),n=n()) %>%
```

```
  ggplot(aes(x=reorder(Rating,sales), y=sales)) +
  geom_segment( aes(x=reorder(Rating,sales), xend=reorder(Rating,sales),
                    y=0, yend=sales), color="black") +
  geom_point( color="blue", size=4) +
  geom_text(aes(label=paste("#games:",n)),hjust=0.9, vjust=1.75) +
  coord_flip() +
  theme_bw() +
  labs(x="Rating", y ="Global Sales per title [M units]",
       title="Global Sales per one title per game rating")

# Excluding Rating NA values from data set (as they can't be imputed with median)
video_sales <- video_sales %>% filter(!is.na(Rating))


# 3.6. Game title analysis ------------------------------------------------


# 3.6.1. Creating a wordcloud ---------------------------------------------

# Creating text corpus from Name string
text <- Corpus(VectorSource(video_sales$Name))

# Cleaning Name string
text <- text %>%
  tm_map(removeNumbers) %>%
  tm_map(removePunctuation) %>%
  tm_map(stripWhitespace) %>%
  tm_map(removeWords, stopwords("english"))

# Changing the format of the data
text <- as.matrix(TermDocumentMatrix(text))

# Sum count of words in text, rearrange  and change into data frame
words <- sort(rowSums(text),decreasing=TRUE)
text.df <- data.frame(word = names(words),freq=words)

# Inspecting new data frame
text.df %>% head(10)

# Excluding "the" as word
text.df <- text.df[!text.df$word=="the",]

# Creating wordcloud of cleaned Name string
wordcloud(text.df$word, text.df$freq, min.freq = 50, max.words=200,
          random.order=FALSE, max.freq=400, rot.per=0.35,
          colors=brewer.pal(12, "Paired"))


# 3.6.2. Checking correlation between words and Global Sales --------------

# Creating a sum sales column to each word in text.df
c <- count(text.df)[[1]]
for (i in 1:c) {
```

```
  text.df$sales[i] <-
    video_sales$Global_Sales[str_detect(video_sales$Name,
                                        fixed(text.df$word[i],
                                              ignore_case=TRUE))] %>%
    sum()
}

# Global Sales vs frequency of the word plot
text.df %>%
  ggplot(aes(x=freq, y=sales)) +
  geom_point() +
  geom_smooth(method=loess, se=TRUE) +
  theme_bw() +
  labs(x="Frequency of the word used in game title",
       y ="Global Sales [M units]",
       title="Sales per frequency of words used in the game title")

# Removing Name variable from video_sales data set
video_sales <- video_sales %>% select(-Name)

# Cleaning the environment
rm(text.df, text, c, i, words)


# 3.7. Final check of the data set ----------------------------------------

# Checking out medians and outliers
summary(video_sales)

# Checking if there are any NAs left
is.na(video_sales) %>% sum()

# Checking out the status
status(video_sales)



# 4. Modeling ------------------------------------------------------------


# 4.1. Correlation between variables and PCA ------------------------------

# Correlation plot of variables in video_sales data set
corrgram(video_sales[,sapply(video_sales, is.numeric)],
         order=TRUE, lower.panel=panel.shade,
         upper.panel=panel.cor, text.panel=panel.txt,
         main="Correlogram of variables selected for modeling")


# Creating a dataframe with only numerical values
# from *video_sales* to use in PCA analysis
pca_data <- video_sales %>% select(-Platform,-Genre,-Rating)
```

```r
# PCA calculations
pr.vsales <- prcomp(pca_data, scale=TRUE)
summary(pr.vsales)

# Variance and cumulative variance plots
pr.var <- pr.vsales$sdev^2
pve <- pr.var/sum(pr.var)
par(mfrow=c(2,1))
plot(pve, type="b", lwd=2, col="#FD8D3C",
     main="Variance of PCA",
     xlab="Number of Principal Components",
     ylab="% of Variance")
plot(cumsum(pve), type="b", lwd=2, col="#41B6C4",
     main="Cumulative variance of PCA",
     xlab="Number of Principal Components",
     ylab="% of Cumulative Variance")
abline(h=0.9, lty=2)

# Cleaning the environment
rm(pca_data,pr.vsales,pr.var,pve)


# 4.2. Global Sales prediction models and their results --------------------


# 4.2.1. Creating a training and testing sub-sets --------------------------

# Removed the regional sales variables, as they make Global Sales 1,
# and we don't want to use them as predictors
globalsales <- video_sales %>%
  select(-NA_Sales,-EU_Sales,-Other_Sales, -JP_Sales)

# Setting seed for reproducibility
set.seed(1)

# Creating train and test sets following 80/20 Pareto Principle
global.test_index <- createDataPartition(y = globalsales$Global_Sales,
                                         times = 1, p = 0.2, list = FALSE)
global.test <- globalsales %>% slice(global.test_index)
global.train <- globalsales %>% slice(-global.test_index)

# Making sure that both test and train have same set of data
global.train <- global.train %>%
  semi_join(global.test, by = "Platform") %>%
  semi_join(global.test, by="Genre") %>%
  semi_join(global.test, by="Rating")


# 4.2.2. Models for predicting Global Sales -------------------------------

# Linear regression model (10-fold cv, pre-processed)
global.lm.fit <- train(Global_Sales~.,data=global.train, method="lm",
                       trControl=trainControl(method="cv",number=10),
```

```
                        preProcess = c("center", "scale"))

global.lm.pred <- predict(global.lm.fit,global.test)

# Cubist (10-fold cv, pre-processed), with tuned committees and neighbors
global.cubist.fit <- train(Global_Sales~.,data=global.train, method="cubist",
                           trControl=trainControl(method="cv",number=10),
                           tuneGrid=data.frame(committees=seq(1,9,1),
                                               neighbors=seq(1,9,1)),
                           preProcess = c("center", "scale"))

global.cubist.pred <- predict(global.cubist.fit,global.test)

plot(global.cubist.fit,highlight = TRUE)
global.cubist.fit$bestTune

# Regression trees (10-fold cv, pre-processed), with tuned cp
global.rpart.fit <- train(Global_Sales~.,data=global.train, method="rpart",
                          tuneGrid=data.frame(cp=seq(0,0.05,0.005)),
                          trControl=trainControl(method="cv",number=10),
                          preProcess = c("center", "scale"))

global.rpart.pred <- predict(global.rpart.fit,global.test)

ggplot(global.rpart.fit, highlight=TRUE)
global.rpart.fit$bestTune

# Random forest (10-fold cv, pre-processed), with 100 trees and tuned mtry
global.rf.fit <- train(Global_Sales~., data = global.train, method="rf",
                                    trControl=trainControl(method="cv", number=10),
                                    tuneGrid=data.frame(mtry=seq(1,25,1)),
                                    ntree=100, preProcess = c("center", "scale"))

global.rf.pred <- predict(global.rf.fit,global.test)

global.rf.fit$bestTune
ggplot(global.rf.fit, highlight=TRUE)

# Non-negative least squares NNLS (10-fold cv, pre-processed)
global.nnls.fit <- train(Global_Sales~.,data=global.train, method="nnls",
                         trControl=trainControl(method="cv",number=10),
                         preProcess = c("center", "scale"))

global.nnls.pred <- predict(global.nnls.fit,global.test)


# 4.2.3. Predicting Global Sales results ----------------------------------

# Creating a list of fitted models, and plotting RMSE, MAE and R-squared results
models <- list(lm=global.lm.fit,cubist=global.cubist.fit,rpart=global.rpart.fit,
               rf=global.rf.fit,nnls=global.nnls.fit)

bwplot(resamples(models),metric=c("RMSE","Rsquared","MAE"),
```

```
       main="Global Sales learning models comparison")

# Creating a data frame out of results for fitted models
mae <- list(median(global.lm.fit$resample$MAE),
            median(global.cubist.fit$resample$MAE),
            median(global.rpart.fit$resample$MAE),
            median(global.rf.fit$resample$MAE),
            median(global.nnls.fit$resample$MAE))
rmse <- list(median(global.lm.fit$resample$RMSE),
             median(global.cubist.fit$resample$RMSE),
             median(global.rpart.fit$resample$RMSE),
             median(global.rf.fit$resample$RMSE),
             median(global.nnls.fit$resample$RMSE))
rsqr <- list(median(global.lm.fit$resample$Rsquared),
             median(global.cubist.fit$resample$Rsquared),
             median(global.rpart.fit$resample$Rsquared),
             median(global.rf.fit$resample$Rsquared),
             median(global.nnls.fit$resample$Rsquared))

global.results <- data.frame(Model=c("LM","Cubist","Rpart","RandomForest","NNLS"),
                             Median_MAE=unlist(mae), Median_RMSE=unlist(rmse),
                             Median_Rsquared=unlist(rsqr))
global.results

# Actual vs Predicted output plots for all models
plot.lm <- data.frame(Actuals=global.test$Global_Sales,
                      Predicted=global.lm.pred) %>% mutate(Type="LM")
plot.cubist <- data.frame(Actuals=global.test$Global_Sales,
                          Predicted=global.cubist.pred) %>% mutate(Type="Cubist")
plot.rpart <- data.frame(Actuals=global.test$Global_Sales,
                         Predicted=global.rpart.pred) %>% mutate(Type="Rpart")
plot.rf <- data.frame(Actuals=global.test$Global_Sales,
                      Predicted=global.rf.pred) %>% mutate(Type="RandomForest")
plot.nnls <- data.frame(Actuals=global.test$Global_Sales,
                        Predicted=global.nnls.pred) %>% mutate(Type="NNLS")

final.plot <- rbind(plot.lm,plot.cubist,plot.rpart,plot.rf,plot.nnls)

ggplot(final.plot, aes(x=Predicted, y=Actuals)) +
  geom_point() +
  geom_smooth(method="loess", se=TRUE) +
  facet_wrap(.~Type) +
  theme_bw()

# Cleaning the environment
rm(final.plot,mae,models,rmse,rsqr, plot.cubist,plot.lm,plot.nnls,plot.rf,plot.rpart)


# 4.3. NA Sales prediction models and their results ----------------------


# 4.3.1. Creating a training and testing sub-sets -------------------------
```

```r
# Creating a new data frame for modeling of NA_Sales output.
# Global Sales removed as the variable that we should not know
# (all sales shares are equal to 1)
NAsales <- video_sales %>% select(-Global_Sales)

# Setting seed for reproducibility
set.seed(1)

# Creating train and test sets following 80/20 Pareto Principle
na.test_index <- createDataPartition(y = NAsales$NA_Sales,
                                     times = 1, p = 0.2, list = FALSE)
na.test <- NAsales%>% slice(na.test_index)
na.train <- NAsales %>% slice(-na.test_index)

# Making sure that both test and train have same set of data
na.train <- na.train %>%
  semi_join(na.test, by="Rating") %>%
  semi_join(na.test, by="Platform") %>%
  semi_join(na.test, by="Genre")


# 4.3.2. Models for predicting NA Sales -----------------------------------

# Linear regression model (10-fold cv, pre-processed)
na.lm.fit <- train(NA_Sales~.,data=na.train, method="lm",
                   trControl=trainControl(method="cv",number=10),
                   preProcess = c("center", "scale"))

na.lm.pred <- predict(na.lm.fit,na.test)

# Cubist (10-fold cv, pre-processed), with tuned committees and neighbors parameter
na.cubist.fit <- train(NA_Sales~.,data=na.train, method="cubist",
                       trControl=trainControl(method="cv",number=10),
                       tuneGrid=data.frame(committees=seq(1,9,1),
                                           neighbors=seq(1,9,1)),
                       preProcess = c("center", "scale"))

na.cubist.pred <- predict(na.cubist.fit,na.test)

plot(na.cubist.fit,highlight = TRUE)
na.cubist.fit$bestTune

# Rpart (10-fold cv, pre-processed), with tuned cp parameter
na.rpart.fit <- train(NA_Sales~.,data=na.train, method="rpart",
                      tuneGrid=data.frame(cp=seq(0,0.05,0.005)),
                      trControl=trainControl(method="cv",number=10),
                      preProcess = c("center", "scale"))

na.rpart.pred <- predict(na.rpart.fit,na.test)

ggplot(na.rpart.fit,highlight = TRUE)
na.rpart.fit$bestTune
```

```r
# Random Forest (10-fold cv, pre-processed), with 100 trees and tunable mtry parameter
na.rf.fit <- train(NA_Sales~.,data=na.train, method="rf",
                   tuneGrid=data.frame(mtry=seq(1,25,1)), ntree=100,
                   trControl=trainControl(method="cv",number=10),
                   preProcess = c("center", "scale"))

na.rf.pred <- predict(na.rf.fit,na.test)

ggplot(na.rf.fit,highlight = TRUE)
na.rf.fit$bestTune

# Non-negative least squares NNLS (10-fold cv, pre-processed)
na.nnls.fit <- train(NA_Sales~.,data=na.train, method="nnls",
                     trControl=trainControl(method="cv",number=10),
                     preProcess = c("center", "scale"))

na.nnls.pred <- predict(na.nnls.fit,na.test)


# 4.3.3. Predicting NA Sales results ------------------------------------

# Creating a list of fitted models, and plotting RMSE, MAE and R-squared results
models <- list(lm=na.lm.fit, cubist=na.cubist.fit, rpart=na.rpart.fit,
               randomforest=na.rf.fit, nnls=na.nnls.fit)

bwplot(resamples(models),metric=c("RMSE","Rsquared","MAE"),
       main="NA Sales learning models comparison")

# Creating a data frame out of results for fitted models
mae <- list(median(na.lm.fit$resample$MAE),
            median(na.cubist.fit$resample$MAE),
            median(na.rpart.fit$resample$MAE),
            median(na.rf.fit$resample$MAE),
            median(na.nnls.fit$resample$MAE))
rmse <- list(median(na.lm.fit$resample$RMSE),
             median(na.cubist.fit$resample$RMSE),
             median(na.rpart.fit$resample$RMSE),
             median(na.rf.fit$resample$RMSE),
             median(na.nnls.fit$resample$RMSE))
rsqr <- list(median(na.lm.fit$resample$Rsquared),
             median(na.cubist.fit$resample$Rsquared),
             median(na.rpart.fit$resample$Rsquared),
             median(na.rf.fit$resample$Rsquared),
             median(na.nnls.fit$resample$Rsquared))

na.results <- data.frame(Model=c("LM","Cubist","Rpart","RandomForest","NNLS"),
                         Median_MAE=unlist(mae), Median_RMSE=unlist(rmse),
                         Median_Rsquared=unlist(rsqr))
na.results

# Comparison of models results - Actual vs Predicted plots
plot.lm <- data.frame(Actuals=na.test$NA_Sales,
                      Predicted=na.lm.pred) %>% mutate(Type="LM")
```

```
plot.cubist <- data.frame(Actuals=na.test$NA_Sales,
                          Predicted=na.cubist.pred) %>% mutate(Type="Cubist")
plot.rpart <- data.frame(Actuals=na.test$NA_Sales,
                         Predicted=na.rpart.pred) %>% mutate(Type="Rpart")
plot.rf <- data.frame(Actuals=na.test$NA_Sales,
                      Predicted=na.rf.pred) %>% mutate(Type="RandomForest")
plot.nnls <- data.frame(Actuals=na.test$NA_Sales,
                        Predicted=na.nnls.pred) %>% mutate(Type="NNLS")

final.plot <- rbind(plot.lm,plot.cubist,plot.rpart,plot.rf,plot.nnls)

ggplot(final.plot, aes(x=Predicted, y=Actuals)) +
  geom_point() +
  geom_smooth(method="loess", se=TRUE) +
  facet_wrap(.~Type) + theme_bw()

# Variable importance table (top 10), arranged side-by-side
na.lm.imp <- varImp(na.lm.fit)
na.cubist.imp <- varImp(na.cubist.fit)
na.rpart.imp <- varImp(na.rpart.fit)
na.rf.imp <- varImp(na.rf.fit)
na.nnls.imp <- varImp(na.nnls.fit)


imp.lm <- data.frame(LM.Variable=rownames(arrange(na.lm.imp$importance,
                                    desc(Overall)))[1:10],
                     Overall=arrange(na.lm.imp$importance,
                                     desc(Overall))[1:10,])
imp.cubist <- data.frame(Cubist.Variable=rownames(arrange(na.cubist.imp$importance,
                                                  desc(Overall)))[1:10],
                         Overall=arrange(na.cubist.imp$importance,
                                         desc(Overall))[1:10,])
imp.rpart <- data.frame(Rpart.Variable=rownames(arrange(na.rpart.imp$importance,
                                                desc(Overall)))[1:10],
                        Overall=arrange(na.rpart.imp$importance,
                                        desc(Overall))[1:10,])
imp.rf <- data.frame(RF.Variable=rownames(arrange(na.rf.imp$importance,
                                          desc(Overall)))[1:10],
                     Overall=arrange(na.rf.imp$importance,
                                     desc(Overall))[1:10,])
imp.nnls <- data.frame(NNLS.Variable=rownames(arrange(na.nnls.imp$importance,
                                              desc(Overall)))[1:10],
                       Overall=arrange(na.nnls.imp$importance,
                                       desc(Overall))[1:10,])

grid.arrange(tableGrob(imp.cubist),tableGrob(imp.rf),nrow=1)
grid.arrange(tableGrob(imp.lm),tableGrob(imp.rpart),tableGrob(imp.nnls),nrow=1)

# Cleaning the environment
rm(final.plot,mae,models,rmse,rsqr, plot.cubist,plot.lm,plot.nnls,plot.rf,plot.rpart)
```