Video Game Sales Prediction

Maggie Yim

8/31/2021

HarvardX Data Science Capstone Project: Video Game Sales Prediction

Section 1. Introduction

Sales forecasting is important in business as it can enhance the firm to access the trend in the future, improve the decision making so as to increase revenues.

In this project, a sales prediction model is created using the open-source dataset from Kaggle. This model aims to use historical sales data of video game to discover the variables affecting the profit in video game industry and foresee the sales in the future. It involves several machine learning techniques and algorithms such as regression model and regularization.

The dataset contains more than 100,000 video games with sales. Each row represents individual video game and there are 16 corresponding variables, for example, Name, Platform, Year_of_Release, Genre, Developer. Unfortunately, there are missing values for certain variables as this dataset was created by two separate web scrapping and only around 6,900 observations with full information. You can find the full dataset here.

Before building a sales prediction model, the above dataset is inspected to understand each variable, identify if data cleaning is required and gain better insight to determine the machine learning techniques to be used. After that, the data is split into two subsets - training set and validation set. A linear regression model is then built to predict the future profit in the validation set by using the input in training set, considering global average, variation in the following variables. Moreover, regularization is used to improve the model performance.

1. Platform
2. Genre
3. Publisher
4. Developer
5. Year_of_Release

In this document, it will split into four sections. First, it provides the details of the dataset and the insight gained from data exploration and visualization. Second, it shows the modeling process and relevant techniques used in this project. Then, it presents the modeling result and discusses the model performance. Finally, it gives a brief summary of the report, its limitations and future work.

Section 2. Dataset

Section 2.1. Installing Packpage

First, certain R packpages, which would be used in data preparation, analysis and visualization, need to be installed in R. Caret package is used, same as other machine learning algorithms using this code:

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

```
## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

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

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

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

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

```
## Loading required package: ggpubr
```

```r
library(tidyverse)
library(caret)
library(data.table)
library(ggpubr)
```

Section 2.2 Dataset Overview

In this section, some insights are gained from the raw dataset. It can be found via the following URL: https://raw.githubusercontent.com/Maggieykw/edX_Video-Game-Sales-with-Ratings/main/Video_Games_Sales_as_at_22_Dec_2016.csv

The dataset contains 16,719 rows and 16 columns, involving the video games released between 1980 and 2020.

```r
raw_data <-
  read.csv("https://raw.githubusercontent.com/Maggieykw/edX_Video-Game-Sales-with-Ratings/main/Video_Gam
```

Section 2.2.1 General Information of Variables

The dataset contains 16,719 observations with 16 variables. Each observation represents sales information of a video game. Below is the list of the variables with their data type and description. *Name* is designator which is used to identify individuals in dataset. There are 4 qualitative variables - *Platform*, *Genre*, *Publisher* and *Developer*. For the rest of the fields, they are quantitative variables. In the next section, it will going to predict one of the quantitative variable - *Global_Sales*, so called target variable. Both quantitative and qualitative data can be used as both of them can provide different information during modeling and they are often used together to get a full picture of prediction.

```r
#Statistics in the dataset
#List of variables in Dataset
str(raw_data)
```

```
## 'data.frame':    16719 obs. of  16 variables:
##  $ Name           : chr  "Wii Sports" "Super Mario Bros." "Mario Kart Wii" "Wii Sports Resort" ...
##  $ Platform       : chr  "Wii" "NES" "Wii" "Wii" ...
##  $ Year_of_Release: chr  "2006" "1985" "2008" "2009" ...
##  $ Genre          : chr  "Sports" "Platform" "Racing" "Sports" ...
##  $ Publisher      : chr  "Nintendo" "Nintendo" "Nintendo" "Nintendo" ...
##  $ NA_Sales       : num  41.4 29.1 15.7 15.6 11.3 ...
##  $ EU_Sales       : num  28.96 3.58 12.76 10.93 8.89 ...
##  $ JP_Sales       : num  3.77 6.81 3.79 3.28 10.22 ...
##  $ Other_Sales    : num  8.45 0.77 3.29 2.95 1 0.58 2.88 2.84 2.24 0.47 ...
##  $ Global_Sales   : num  82.5 40.2 35.5 32.8 31.4 ...
##  $ Critic_Score   : int  76 NA 82 80 NA NA 89 58 87 NA ...
##  $ Critic_Count   : int  51 NA 73 73 NA NA 65 41 80 NA ...
##  $ User_Score     : chr  "8" "" "8.3" "8" ...
##  $ User_Count     : int  322 NA 709 192 NA NA 431 129 594 NA ...
##  $ Developer      : chr  "Nintendo" "" "Nintendo" "Nintendo" ...
##  $ Rating         : chr  "E" "" "E" "E" ...
```

```r
#Description of Variables
Var_Details <- data_frame("Item"=c(1:16),"Variable" = colnames(raw_data),
                          "Type of Data" = c("Designator", "Qualitative", "Quantitative", "Qualitative"
                          "Description" = c("Game name", "Platform of the game release", "Year of the ga
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
Var_Details %>% knitr::kable()
```

Item	Variable	Type of Data	Description
1	Name	Designator	Game name
2	Platform	Qualitative	Platform of the game release
3	Year_of_Release	Quantitative	Year of the game's release
4	Genre	Qualitative	Genre of the game
5	Publisher	Qualitative	Publisher of the game
6	NA_Sales	Quantitative	Sales in North America (in millions)
7	EU_Sales	Quantitative	Sales in Europe (in millions)
8	JP_Sales	Quantitative	Sales in Japan (in millions)
9	Other_Sales	Quantitative	Sales in the rest of the world (in millions)
10	Global_Sales	Quantitative	Total worldwide sales
11	Critic_Score	Quantitative	Aggregate score compiled by Metacritic staff
12	Critic_Count	Quantitative	Number of critics used in coming up with the Criticscore
13	User_Score	Quantitative	Score by Metacritic's subscribers
14	User_Count	Quantitative	Number of users who gave the userscore
15	Developer	Qualitative	Party responsible for creating the game
16	Rating	Quantitative	Entertainment Software Rating Board (ESRB) ratings

Section 2.2.2 Qualitative Variables

Below lists out the number of unique type in each qualitative variables. All of the variables have different unique type and the sales can then be visualized by those various types to gain additional insight.

```
nPlatform <- n_distinct(raw_data$Platform)
nGenre <- n_distinct(raw_data$Genre)
nPublisher <- n_distinct(raw_data$Publisher)
nDeveloper <- n_distinct(raw_data$Developer)

tibble(Variable = c("Platform", "Genre", "Publisher", "Developer"),
       "Number of Type" = c(nPlatform,nGenre,nPublisher,nDeveloper)) %>%
knitr::kable()
```

Variable	Number of Type
Platform	31
Genre	13
Publisher	582
Developer	1697

Section 2.2.3 Missing Value in Variables

Before developing a predictive model, it is good to check if the data inside the dataset is clean for modeling. There are missing values in the dataset as the data were collected by two different web scraping. You can see the number of missing value (either NA, N/A or) from the following table. In the data visualization part and modeling (i.e. *Section 2.2.4* and *Section 3*), those missing values will be removed beforehand in order to have useful data insights and enhance the reliability of analysis result.

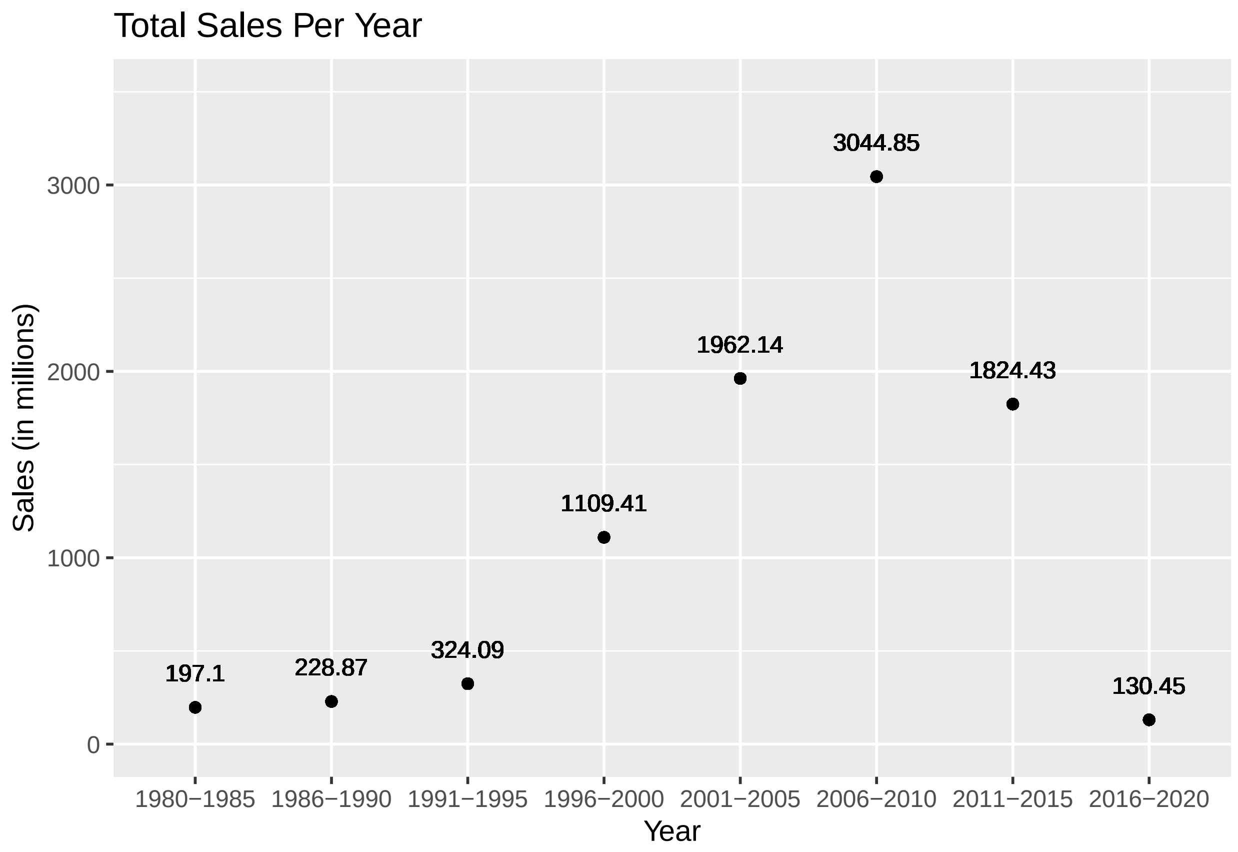
```r
MissingValue <- raw_data %>%
                  mutate(across(everything(),
                                ~ifelse(.=="", NA,
                                 ifelse(.=="N/A",NA,as.character(.))))) %>%
                  summarise_all(~ sum(is.na(.)))

MissingValueCount <- data_frame(
                  "Variable" = colnames(MissingValue),
                  "Number of Missing Value" = t(MissingValue))

MissingValueCount %>% knitr::kable() #result testing
```

Variable	Number of Missing Value
Name	2
Platform	0
Year_of_Release	269
Genre	2
Publisher	54
NA_Sales	0
EU_Sales	0
JP_Sales	0
Other_Sales	0
Global_Sales	0
Critic_Score	8582
Critic_Count	8582
User_Score	6704
User_Count	9129
Developer	6623
Rating	6769

Section 2.2.4 Sales by Different Variables

The variable *Year_of_Release* is split into 7 groups for visualization. The graph below shows that the video game sales was first increased smoothly from 1980 to 1995. After that, the sales was raised sharply to 3044.85 millions in 2010. The sales was plummeted to 130.45 millions in 2020.

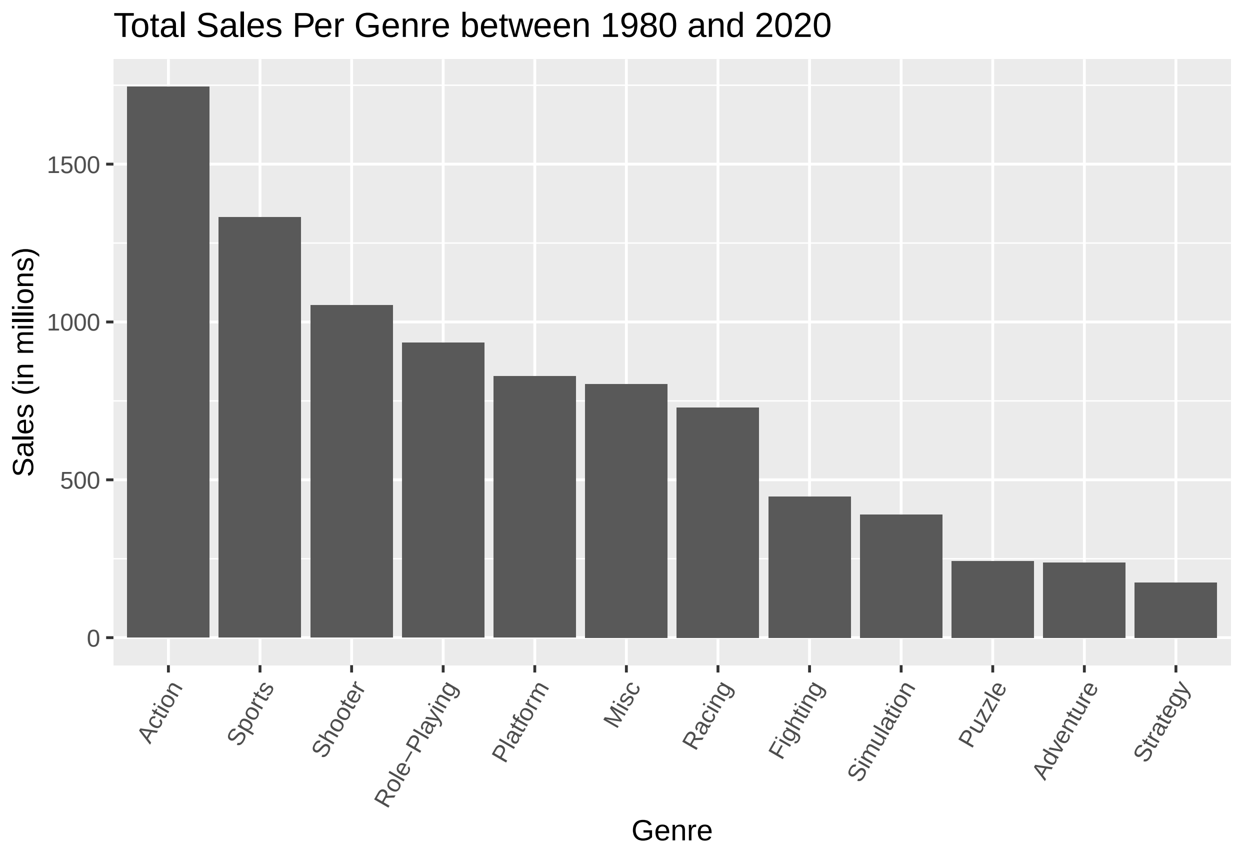
```r
#Graph - Sales per year

raw_data  %>%
  mutate(Year_Range = ifelse(Year_of_Release<=1985, "1980-1985",
                      ifelse(Year_of_Release<=1990, "1986-1990",
                      ifelse(Year_of_Release<=1995, "1991-1995",
                      ifelse(Year_of_Release<=2000, "1996-2000",
                      ifelse(Year_of_Release<=2005, "2001-2005",
                      ifelse(Year_of_Release<=2010, "2006-2010",
                      ifelse(Year_of_Release<=2015, "2011-2015",
                      ifelse(Year_of_Release<=2020, "2016-2020",
                             "Others"))))))))) %>%
  filter(Year_Range != "Others") %>%
  group_by(Year_Range) %>%
  mutate("Total_Sales"=sum(Global_Sales)) %>%
  ggplot(aes(x = Year_Range,
             y = Total_Sales,
             label=Total_Sales)) +
  geom_point() +
  geom_text(position = position_dodge(width = 1),
            vjust = -1.5, size = 3) +
  ylim(0, 3500) +
  ggtitle("Total Sales Per Year") +
  xlab("Year") +
  ylab("Sales (in millions)")
```



There are 13 genre types of video game. Below shows the sales per each genre type. There is significant variation between the genres. For example, in the past 40 years, action video game was the most profitable, generating more than 1,500 million sales while strategy video game was the least profitable, having less than 250 million sales.

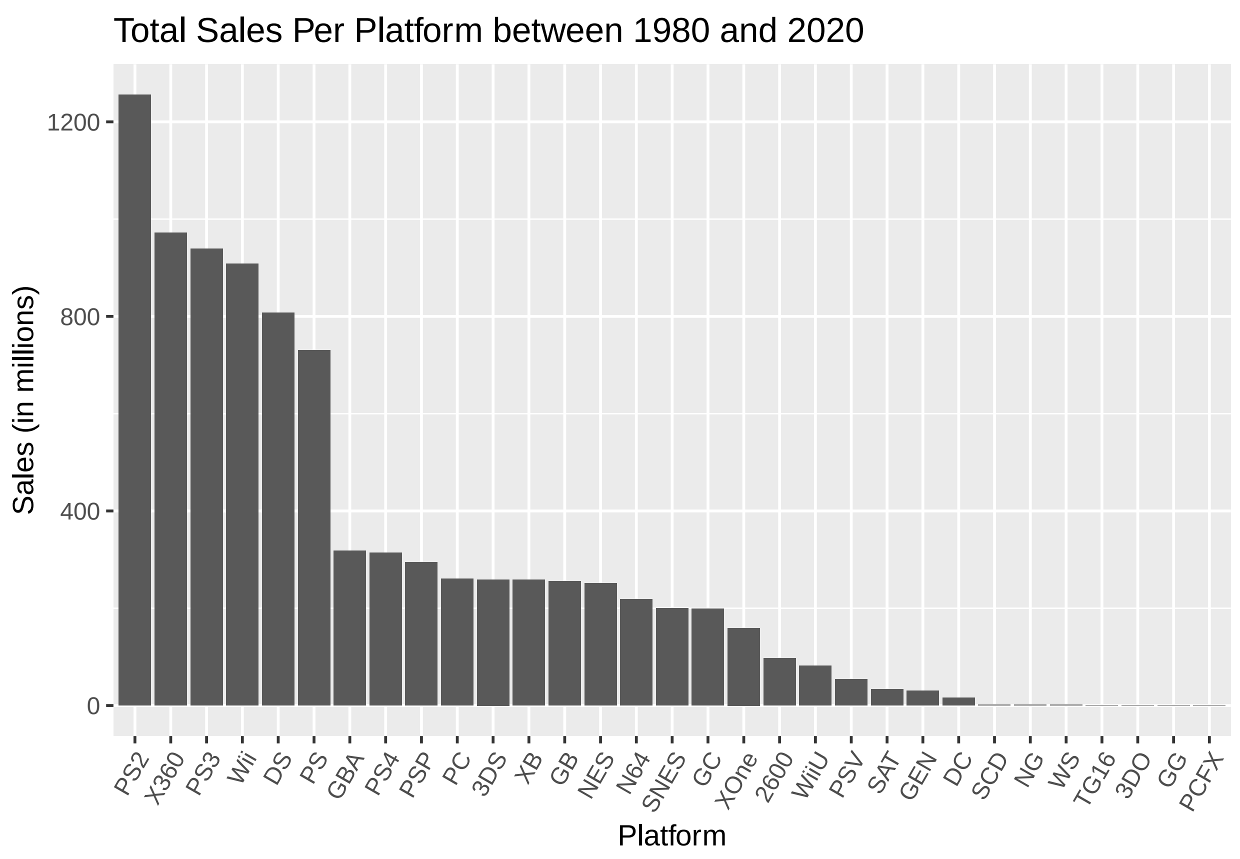
```r
#Graph - Sales per genre

#Data cleaning
adj_raw_data <- raw_data %>% filter(Genre!="")

#New variable to calculate the sales by genre
sum_by_genre <- aggregate(Global_Sales ~ Genre,
                          data = adj_raw_data,
                          FUN = sum)

#Plot graph
```

```r
ggplot(sum_by_genre,
       aes(x = reorder(Genre, -Global_Sales),
           y = Global_Sales))+
  geom_bar(stat = 'identity') +
  ggtitle("Total Sales Per Genre between 1980 and 2020") +
  xlab("Genre") +
  ylab("Sales (in millions)") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1))
```



Below shows the sales per each platform. There are 31 platforms for the video game and a large platform to platform variation is observed. PS2 generated the most sales within 40 years while 4 of the platforms had no sales at all.

```r
#Graph - Sales per platform

#New variable to calculate the sales by platform
sum_by_platform <- aggregate(Global_Sales ~ Platform, data = raw_data, FUN = sum)

#Plot graph
ggplot(sum_by_platform, aes(x = reorder(Platform, -Global_Sales), y = Global_Sales))+
  geom_bar(stat = 'identity') +
  ggtitle("Total Sales Per Platform between 1980 and 2020") +
  xlab("Platform") +
  ylab("Sales (in millions)") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1))
```



Since there is 582 publisher in the dataset (see *Section 2.2.2*), here only shows the top 10 sales the bottom 10 sales of the publishers to ease data exploration. Nintendo was the most profitable publisher, with the sales more than 1,700 millions. All of the bottom 10 sales publishers had the sales equals to 0.01 millions.

```
#Table - Top 10 sales per publisher
raw_data %>%
  group_by(Publisher) %>%
  summarize(Sales_by_publisher = sum(Global_Sales)) %>%
  top_n(10) %>%
  arrange(desc(Sales_by_publisher))%>%
  knitr::kable()
```

```
## Selecting by Sales_by_publisher
```

Publisher	Sales_by_publisher
Nintendo	1788.81
Electronic Arts	1116.96
Activision	731.16
Sony Computer Entertainment	606.48
Ubisoft	471.61
Take-Two Interactive	403.82
THQ	338.44
Konami Digital Entertainment	282.39
Sega	270.35

Publisher	Sales_by_publisher
Namco Bandai Games	254.62

```r
#Table - Bottom 10 sales per publisher
raw_data %>%
  group_by(Publisher) %>%
  summarize(Sales_by_publisher = sum(Global_Sales)) %>%
  top_n(-10) %>%
  arrange(desc(Sales_by_publisher))%>%
  knitr::kable()
```

```
## Selecting by Sales_by_publisher
```

Publisher	Sales_by_publisher
Ascaron Entertainment	0.01
Boost On	0.01
Commseed	0.01
EON Digital Entertainment	0.01
Genterprise	0.01
Interchannel-Holon	0.01
Interworks Unlimited, Inc.	0.01
Inti Creates	0.01
Lighthouse Interactive	0.01
Media Entertainment	0.01
Michaelsoft	0.01
Naxat Soft	0.01
Ongakukan	0.01
Otomate	0.01
Paradox Development	0.01
Piacci	0.01
Red Flagship	0.01
Takuyo	0.01
UIG Entertainment	0.01

Here shows the details top 10 sales and bottom 10 sales of developers as well as there are quite a lot of developers in the dataset. The result is similar to the previous section. Among the developers, sales of Nintendo was the most and the bottom 10 developers had the sales equals to 0.01 millions.

```r
#Table - Top 10 sales per developer
raw_data %>%
  filter(Developer!="") %>%
  group_by(Developer) %>%
  summarize(Sales_by_developer = sum(Global_Sales)) %>%
  top_n(10) %>%
  arrange(desc(Sales_by_developer))%>%
  knitr::kable()
```

```
## Selecting by Sales_by_developer
```

Developer	Sales_by_developer
Nintendo	531.71
EA Sports	175.38
EA Canada	142.32
Ubisoft	132.54
Rockstar North	119.47
Capcom	115.71
Ubisoft Montreal	108.31
Treyarch	103.16
EA Tiburon	96.12
Traveller's Tales	79.22

```r
#Table - Bottom 10 sales per developer
raw_data %>%
  filter(Developer!="") %>%
  group_by(Developer) %>%
  summarize(Sales_by_developer = sum(Global_Sales)) %>%
  top_n(-10) %>%
  arrange(desc(Sales_by_developer))%>%
  knitr::kable()
```

```
## Selecting by Sales_by_developer
```

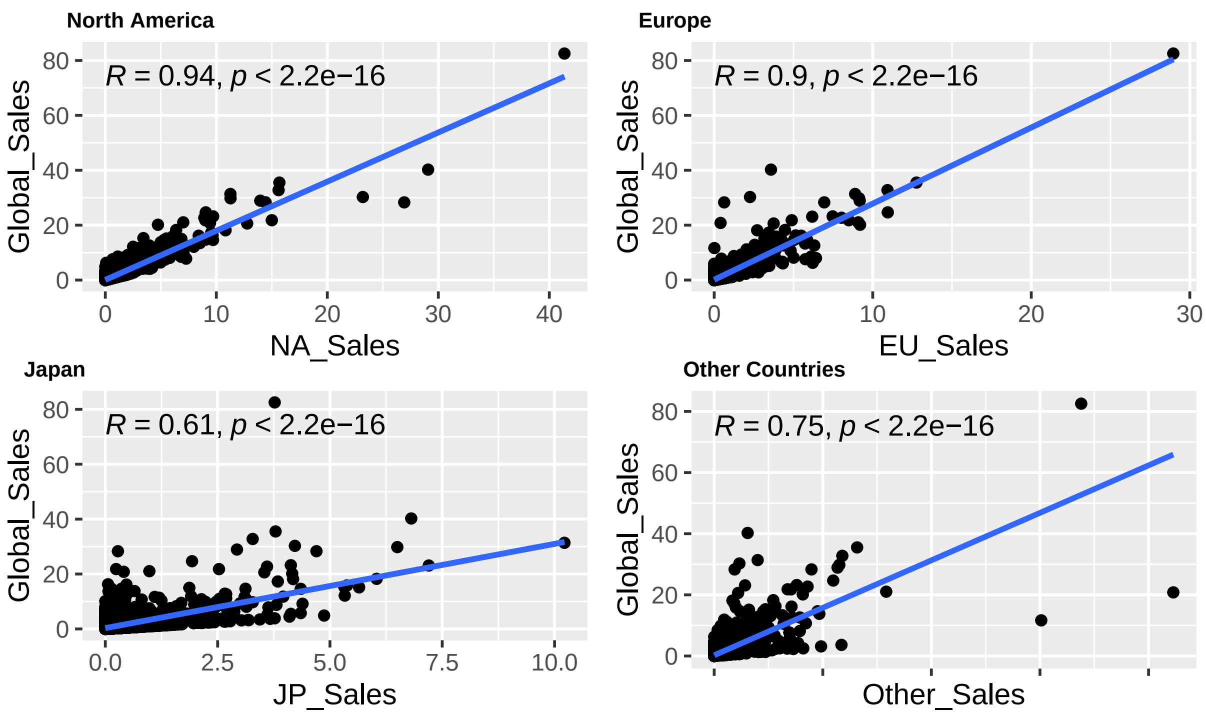
Developer	Sales_by_developer
1C, Various, 1C Company	0.01
777 Studios	0.01
Atomic Games	0.01
Battlefront.com, 1C, 1C Company	0.01
Big Red Software	0.01
Bigben Interactive, Red Wagon Games	0.01
Black Bean Games	0.01
Blue Tea Games	0.01
Boston Animation	0.01
Camouflaj, LLC	0.01
Coffee Stain Studios	0.01
Compulsion Games	0.01
DMA Design, Rockstar North	0.01
dtp Young Entertainment AG	0.01
EA Phenomic	0.01
Empty Clip Studios	0.01
Headgate	0.01
High Moon Studios, Mercenary Technologies	0.01
Inferno Games	0.01
Infinite Dreams, Paragon 5	0.01
Interchannel-Holon	0.01
Interworks Unlimited, Inc.	0.01
Katauri Interactive	0.01
Next Wave Team	0.01
Papyrus	0.01
React Games	0.01

Developer	Sales_by_developer
Societe Pollene	0.01
Spidersoft, Spiders	0.01
Subterranean Games	0.01
TDK Mediactive	0.01
Tecmo, Graphic Research	0.01
TML-Studios	0.01
Triple Eh? Ltd	0.01

The total sales in dataset is composited by the sales in different countries - Norther America, Europe, Japan and the rest of the world. In the following scatter plots, it implies the total sales and sales in different countries have the positive correlation. North America had the high correlation coefficient, which is 0.94 while Japan had the least correlation coefficient, which is 0.61. Second observation is that although all scatter plots show the positive correlation, there are some points that are farthest from the regression line in the graph of Japan and other countries. It implies that those observations lie outside the overall pattern of the distribution.

```r
#Graph - Different sales correlation

NA_plot <- ggplot(raw_data, aes(x=NA_Sales, y=Global_Sales)) +
  geom_point()+
  geom_smooth(method=lm) +
  geom_smooth(method=lm)+ stat_cor(method = "pearson")

EU_plot <- ggplot(raw_data, aes(x=EU_Sales, y=Global_Sales)) +
  geom_point()+
  geom_smooth(method=lm) +
  geom_smooth(method=lm)+ stat_cor(method = "pearson")

JP_plot <- ggplot(raw_data, aes(x=JP_Sales, y=Global_Sales)) +
  geom_point()+
  geom_smooth(method=lm) +
  geom_smooth(method=lm)+ stat_cor(method = "pearson")

Rest_plot <- ggplot(raw_data, aes(x=Other_Sales, y=Global_Sales)) +
  geom_point() +
  geom_smooth(method=lm)+ stat_cor(method = "pearson")

ggarrange(NA_plot, EU_plot, JP_plot,Rest_plot + rremove("x.text"),
          labels = c("North America", "Europe", "Japan", "Other Countries"),
          ncol = 2, nrow = 2,
          font.label = list(size = 8),
          vjust = 0)+
          theme(plot.margin = margin(0.2,0.2,2,0.2, "cm"))
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



Section 3. Method

Section 3.1. Variable Selection

Recapping the result of *Section 2.2.1* and *Section 2.2.3*, most of the quantitative variables (for example *Critic_Count*, *Critic_Score*, *User_Count*, *User_Score*, etc) have a lot of missing values and therefore the model only took account of the following 5 variables to predict the variable *Global_Sales*:

1. Platform
2. Genre
3. Publisher
4. Developer
5. Year_of_Release

```
Merge1 <- merge(MissingValueCount,Var_Details,by="Variable")
Merge2<- Merge1[c("Type of Data","Variable","Number of Missing Value")]
Merge2[order(Merge2$`Type of Data`),]%>% knitr::kable()
```

	Type of Data	Variable	Number of Missing Value
9	Designator	Name	2
3	Qualitative	Developer	6623
5	Qualitative	Genre	2
11	Qualitative	Platform	0

	Type of Data	Variable	Number of Missing Value
12	Qualitative	Publisher	54
1	Quantitative	Critic_Count	8582
2	Quantitative	Critic_Score	8582
4	Quantitative	EU_Sales	0
6	Quantitative	Global_Sales	0
7	Quantitative	JP_Sales	0
8	Quantitative	NA_Sales	0
10	Quantitative	Other_Sales	0
13	Quantitative	Rating	6769
14	Quantitative	User_Count	9129
15	Quantitative	User_Score	6704
16	Quantitative	Year_of_Release	269

Section 3.2. Data Cleaning

As there is missing value in the dataset, it requires to be cleaned before modeling development. Now the dataset size is reduced from 16,719 to 9,904.

```
# Data Cleaning
raw_data <- raw_data %>%
          filter(Year_of_Release!="N/A") %>%
          filter(Publisher!="N/A") %>%
          filter(Genre!="") %>%
          filter(Developer!="") %>%
          select(Platform,Year_of_Release,Genre,Publisher,Developer,Global_Sales)

nrow(raw_data)
```

```
## [1] 9904
```

Section 3.3. Dataset Split Ratio

To build sales prediction model, the dataset is first split into two sets of data which are training set and variation set with an 70:30 ratio. The training set is used to train the model while the variation set is used to evaluate the model fit on the training set and fine-tune model parameters so as to control machine learning process and enhance the model accuracy.

The decision of dataset split ratio is case by case but it depends on two main things: actual model to be trained and the total data sample in dataset. If there has very few parameters to control the model training, it will be easier to validate and tune the model so the size of validation set can be reduced. In the other hand, if the model has a lot of parameters to be used, it is preferable to have a larger validation set. Generally, dataset is split into training set and validation set with an 80:20 ratio. In case the data size is very large, one also goes for a 90:10 data split ratio where the validation data set represents 10% of the data. In this project, 70% of dataset went to training and the rest 30% is used as validation set because the dataset is relatively small after data cleaning.

```
# Validation set will be 20% of raw data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use ‘set.seed(1)‘
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
test_index <- createDataPartition(y = raw_data$Global_Sales, times = 1, p = 0.3, list = FALSE)
train_set <- raw_data[-test_index,]
temp <- raw_data[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
                semi_join(train_set, by = "Platform") %>%
                semi_join(train_set, by = "Year_of_Release") %>%
                semi_join(train_set, by = "Genre") %>%
                semi_join(train_set, by = "Publisher") %>%
                semi_join(train_set, by = "Developer")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("Platform", "Year_of_Release", "Genre", "Publisher", "Developer", "Global_Sales")
```

```r
train_set <- rbind(train_set, removed)

# Remove temporary variable
rm(test_index, temp, raw_data, removed)
```

Section 3.4. Model Performance Measure - Root Mean Squared Error

In order to compare different model or compare model with some baseline, a loss function is needed to quantify how it perform. There are two common methods to measure the accuracy of a recommendation system: Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE).

MAPE is the average of the absolute differences between actual observation and prediction in term of percentage, without considering the direction and all individual difference has equal weight.

RMSE is the square root of the average of squared difference between actual observation and prediction. In other words, it measures the average magnitude of error. In this project, RMSE is used to evaluate the model performance. Although RMSE does not express mean of the error alone and has implication that is difficult to understand or tease out, RMSE has distinct advantage over MAPE, which is avoiding to use absolute value as it is not desirable in most of the mathematical calculations. Moreover, RMSE can penalize large errors more than MAPE.

The following function is created to compute this residual means squared error for a vector of ratings and their corresponding predictors.

```r
RMSE <- function(true_rating, predicted_rating){ sqrt(mean((true_rating - predicted_rating)^2))
}
```

Section 3.5. Designing Movie Recommendation System

In this project, the effect of each independent variable is considered in modeling by calculating the mean of deviation of each sales within each type of independent variable. In short, the dataset is grouped by each independent variable and the distance between the average sales in each category and the overall average rating is calculated. After that, it has a coefficient for each independent variable. Those coefficients are then added to the overall average sales to perform the sales prediction. The model development will be shown in the following parts.

Section 3.5.1. Considering average in the model only

First, let's start to build the simplest sale prediction model by assuming the same sales for all video game regardless of other information. In order to minimize the RMSE, the assigned rating is the least squares estimate of the "true" rating for all movies. In this case, it is the mean of all sales in training set:

Computation of Average Sales

```
#Considering Global Average in the Model only

mu_hat <- mean(train_set$Global_Sales)
mu_hat
```

```
## [1] 0.5994874
```

RMSE Result

After that, we can test the model performance using validation set. If all unknown sales in validation set is predicted by the above sales average, we obtain the following RMSE. The RMSE of the model including global average of rating is 1.510188, which is quite high because this model is generalizing for video game and not personalizing for each game. It can be improved by adding more variables into the model.

```
naive_rmse <- RMSE(validation$Global_Sales, mu_hat)
rmse_results <- data_frame("Predictive Method" = "Just Global Average", RMSE = naive_rmse)

rmse_results %>% knitr::kable() #result testing
```

Predictive Method	RMSE
Just Global Average	1.510188

Section 3.5.2. Modeling Platform Effect

From the graph in *Section 2.2.4* below, it shows that some platforms generated more sales than others. Therefore, the previous model can be improved by adding additional effect to shorten the difference between the predicted value and the correct value which we are trying to predict.

Computation of Platform Effect Estimate

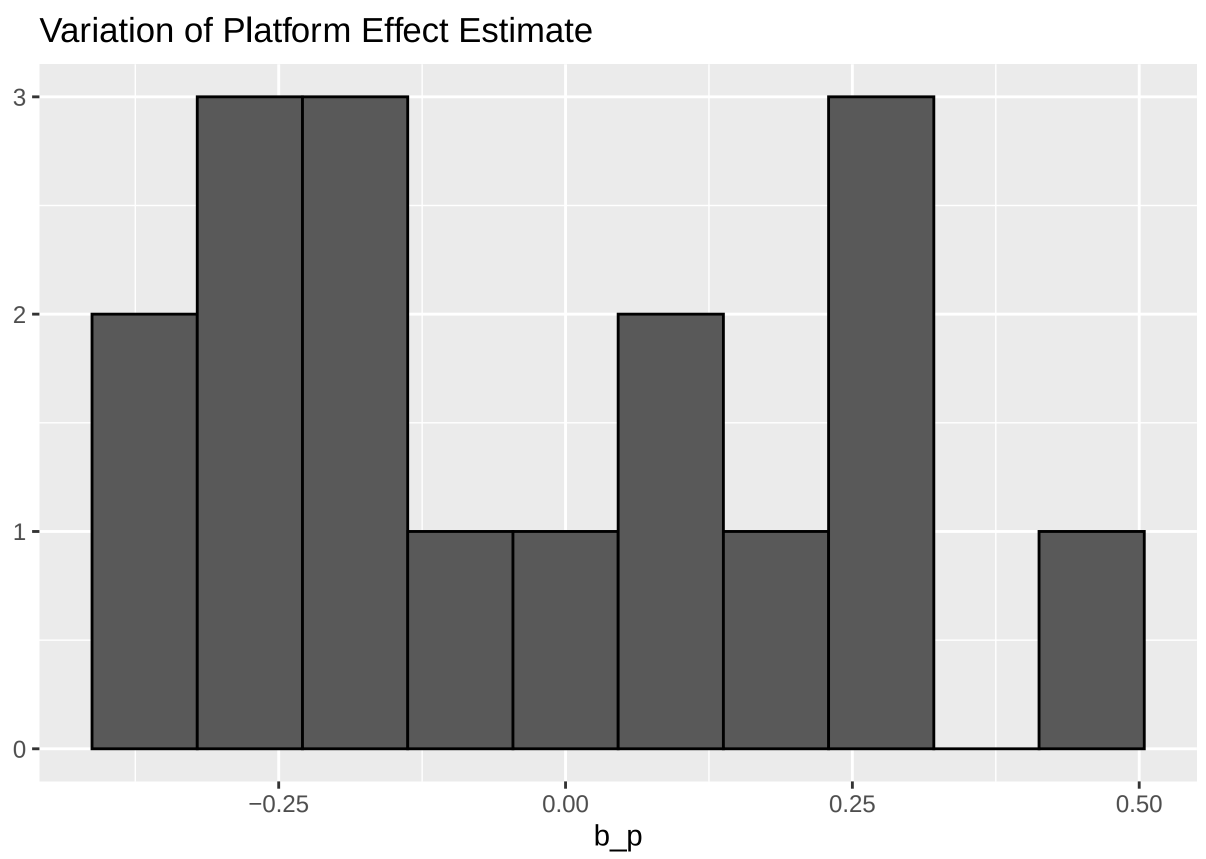
As the least squares estimate effect is just the average of (specific platform's sales - average of all rating), the effect can be computed this way:

```
platform_avgs <- train_set %>%
  group_by(Platform) %>%
  summarize(b_p = mean(Global_Sales - mu_hat))
```

Variation of Platform Effect Estimate

From the graph below, it shows these effect estimates vary substantially.

```
qplot(b_p, data = platform_avgs, bins = 10, color = I("black"))+
  ggtitle("Variation of Platform Effect Estimate")
```



RMSE Result

After the platform effect is added into the predictive model, the RMSE is decreased from 1.510188 to 1.490115. In other words, the model can predict better sales now.

```
#Adding Platform Effect into Model

predicted_sales <- mu_hat + validation %>%
  left_join(platform_avgs, by='Platform') %>%
  .$b_p

model_1_rmse <- RMSE(predicted_sales, validation$Global_Sales)
model_1_rmse #for result testing
```

```
## [1] 1.490115
```

```
rmse_results <- bind_rows(rmse_results,
                  data_frame("Predictive Method"="Platform Effect Model",
                  RMSE = model_1_rmse ))

rmse_results %>% knitr::kable() #for result testing
```

Predictive Method	RMSE
Just Global Average	1.510188

Predictive Method	RMSE
Platform Effect Model	1.490115

Section 3.5.3. Modeling Genre Effect

There is substantial variability across genres as well (see *Section 2.2.4* for details). Some video game genres were more profitable. This implies that the previous model can be further enhanced by adding genre-specific effect as well.

Computation of Genre Effect Estimate

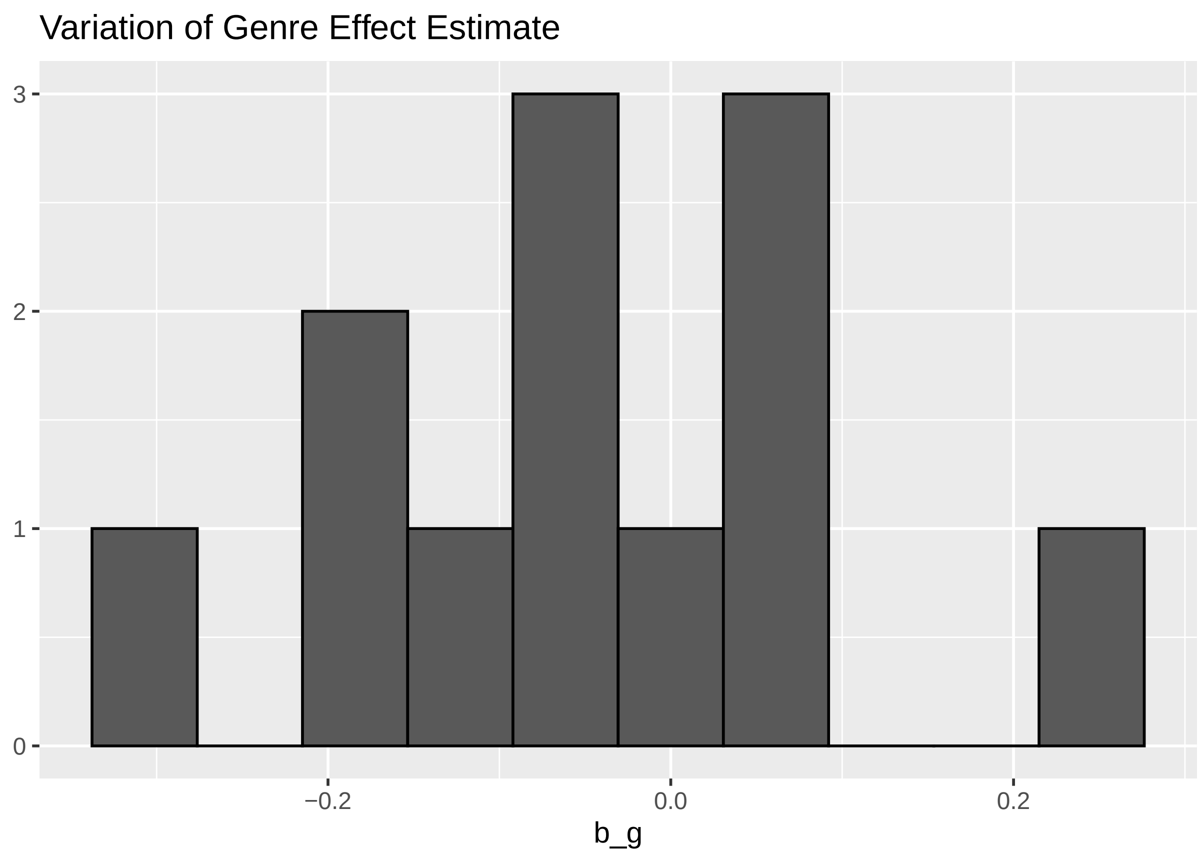
Similar to platform effect, the genre effect can be computed this way:

```
genre_avgs <- train_set %>%
  left_join(platform_avgs, by='Platform') %>%
  group_by(Genre) %>%
  summarize(b_g = mean(Global_Sales - mu_hat - b_p))
```

From the graph below, it shows these effect estimates vary substantially.

Variation of Genre Effect Estimate

```
qplot(b_g, data = genre_avgs, bins = 10, color = I("black"))+
  ggtitle("Variation of Genre Effect Estimate")
```



RMSE Result

After the genre effect is also added into the predictive model, the RMSE is decreased from 1.490115 to 1.489390. The model performance is slightly enhanced.

```
#Adding Genre Effect into Model

predicted_sales <- validation %>%
  left_join(platform_avgs, by='Platform') %>%
  left_join(genre_avgs, by='Genre') %>%
  mutate(pred = mu_hat + b_p + b_g) %>%
  .$pred

model_2_rmse <- RMSE(predicted_sales, validation$Global_Sales)
model_2_rmse #for result testing
```

```
## [1] 1.48939
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Platform + Genre Effects Model",
                                     RMSE = model_2_rmse ))

rmse_results %>% knitr::kable() #for result testing
```

Predictive Method	RMSE
Just Global Average	1.510188
Platform Effect Model	1.490115
Platform + Genre Effects Model	1.489390

Section 3.5.4. Modeling Publisher Effect

The following code will do the similar procedure to add publisher effect into the model.

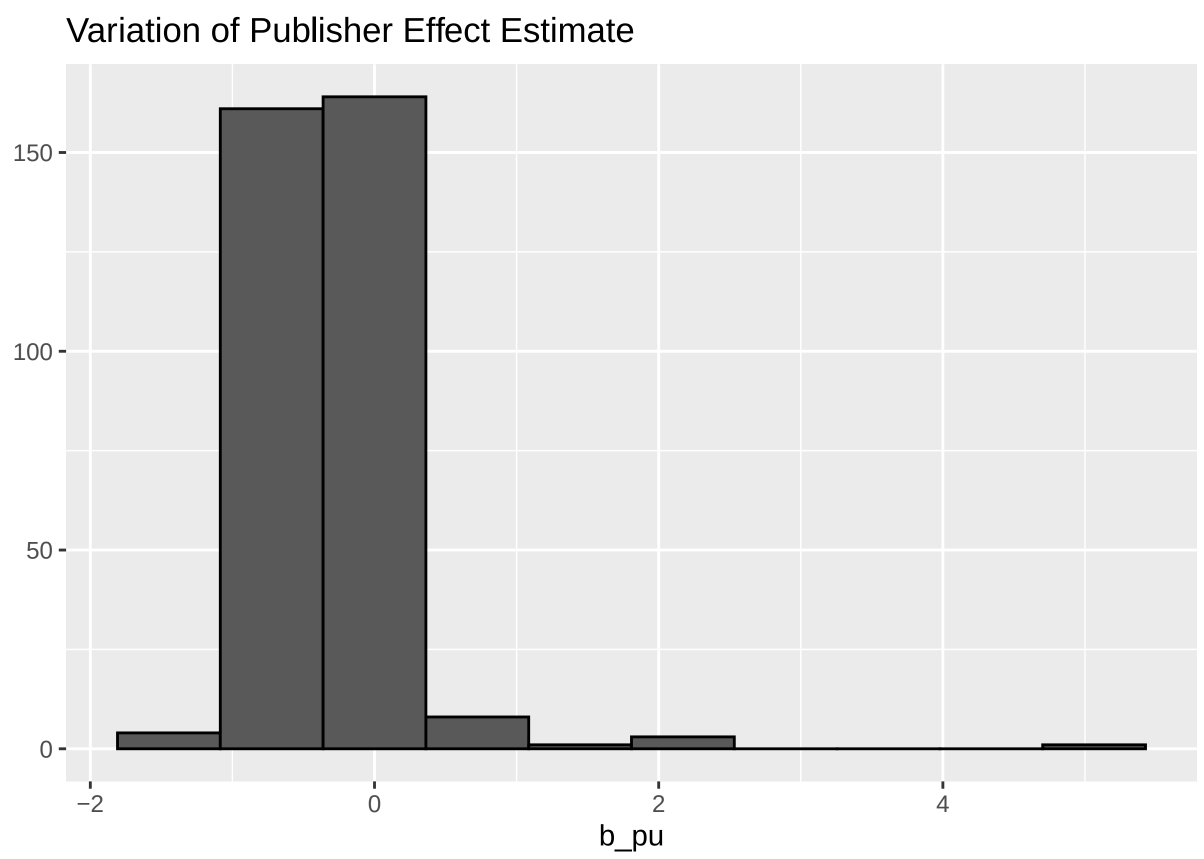
Computation of Publisher Effect Estimate

Publisher effect is calculated as follows:

```
#Adding Publisher Effect into Model

publisher_avgs <- train_set %>%
  left_join(platform_avgs, by='Platform') %>%
  left_join(genre_avgs, by='Genre') %>%
  group_by(Publisher) %>%
  summarize(b_pu = mean(Global_Sales - mu_hat - b_p - b_g))
```

Variation of Publisher Effect Estimate

```
#Variation of Publisher Effect Estimate
qplot(b_pu, data = publisher_avgs, bins = 10, color = I("black"))+
  ggtitle("Variation of Publisher Effect Estimate")
```



RMSE Result

When publisher effect is included in the model, the RMSE is largely dropped from 1.489390 to 1.409233. It implies that publisher effect is significant to affect the video game sales.

```
predicted_sales <- validation %>%
  left_join(platform_avgs, by='Platform') %>%
  left_join(genre_avgs, by='Genre') %>%
  left_join(publisher_avgs, by='Publisher') %>%
  mutate(pred = mu_hat + b_p + b_g + b_pu) %>%
  .$pred

model_3_rmse <- RMSE(predicted_sales, validation$Global_Sales)
model_3_rmse #for result testing
```

```
## [1] 1.409233
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Platform + Genre + Publisher Effects Model",
                                     RMSE = model_3_rmse ))

rmse_results %>% knitr::kable() #for result testing
```

Predictive Method	RMSE
Just Global Average	1.510188
Platform Effect Model	1.490115
Platform + Genre Effects Model	1.489390
Platform + Genre + Publisher Effects Model	1.409233

Section 3.5.5. Modeling Developer Effect

The following code will do the similar procedure to add developer effect into the model.

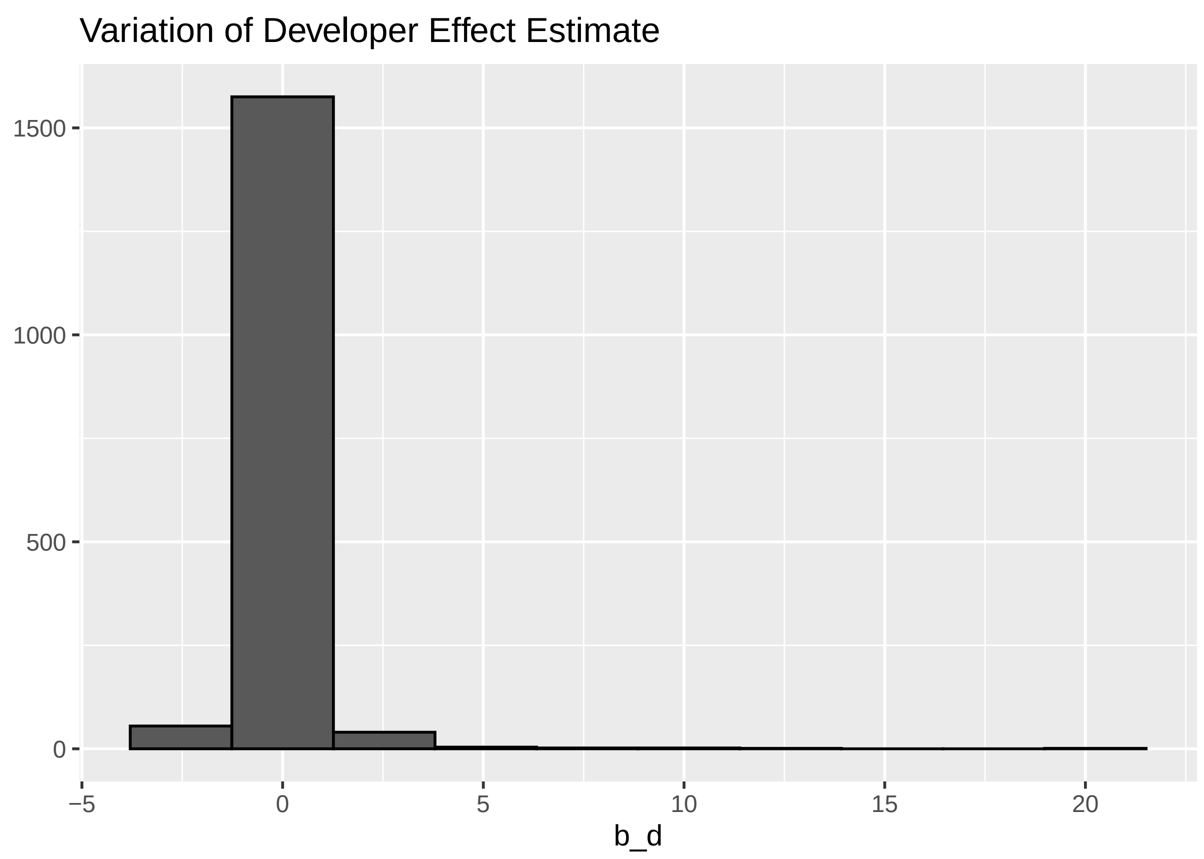
Computation of Developer Effect Estimate

Developer effect is calculated as follows:

```
#Adding Developer Effect into Model

developer_avgs <- train_set %>%
  left_join(platform_avgs, by='Platform') %>%
  left_join(genre_avgs, by='Genre') %>%
  left_join(publisher_avgs, by='Publisher') %>%
  group_by(Developer) %>%
  summarize(b_d = mean(Global_Sales - mu_hat - b_p - b_g - b_pu))
```

Variation of Developer Effect Estimate

```
#Variation of Developer Effect Estimate
qplot(b_d, data = developer_avgs, bins = 10, color = I("black"))+
  ggtitle("Variation of Developer Effect Estimate")
```



RMSE Result

When developer effect is added in the model, the RMSE is dropped from 1.409233 to 1.329279. It implies that developer effect is also important during sales prediction modeling.

```
predicted_sales <- validation %>%
  left_join(platform_avgs, by='Platform') %>%
  left_join(genre_avgs, by='Genre') %>%
  left_join(publisher_avgs, by='Publisher') %>%
  left_join(developer_avgs, by='Developer') %>%
  mutate(pred = mu_hat + b_p + b_g + b_pu + b_d) %>%
  .$pred

model_4_rmse <- RMSE(predicted_sales, validation$Global_Sales)
#model_4_rmse #for result testing

rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Platform + Genre + Publisher + Developer Effe
                                     RMSE = model_4_rmse ))

rmse_results %>% knitr::kable() #for result testing
```

Predictive Method	RMSE
Just Global Average	1.510188
Platform Effect Model	1.490115

Predictive Method	RMSE
Platform + Genre Effects Model	1.489390
Platform + Genre + Publisher Effects Model	1.409233
Platform + Genre + Publisher + Developer Effects Model	1.329279

Section 3.5.6. Modeling Year of Release Effect

The following code will do the similar procedure to add year of release effect into the model.

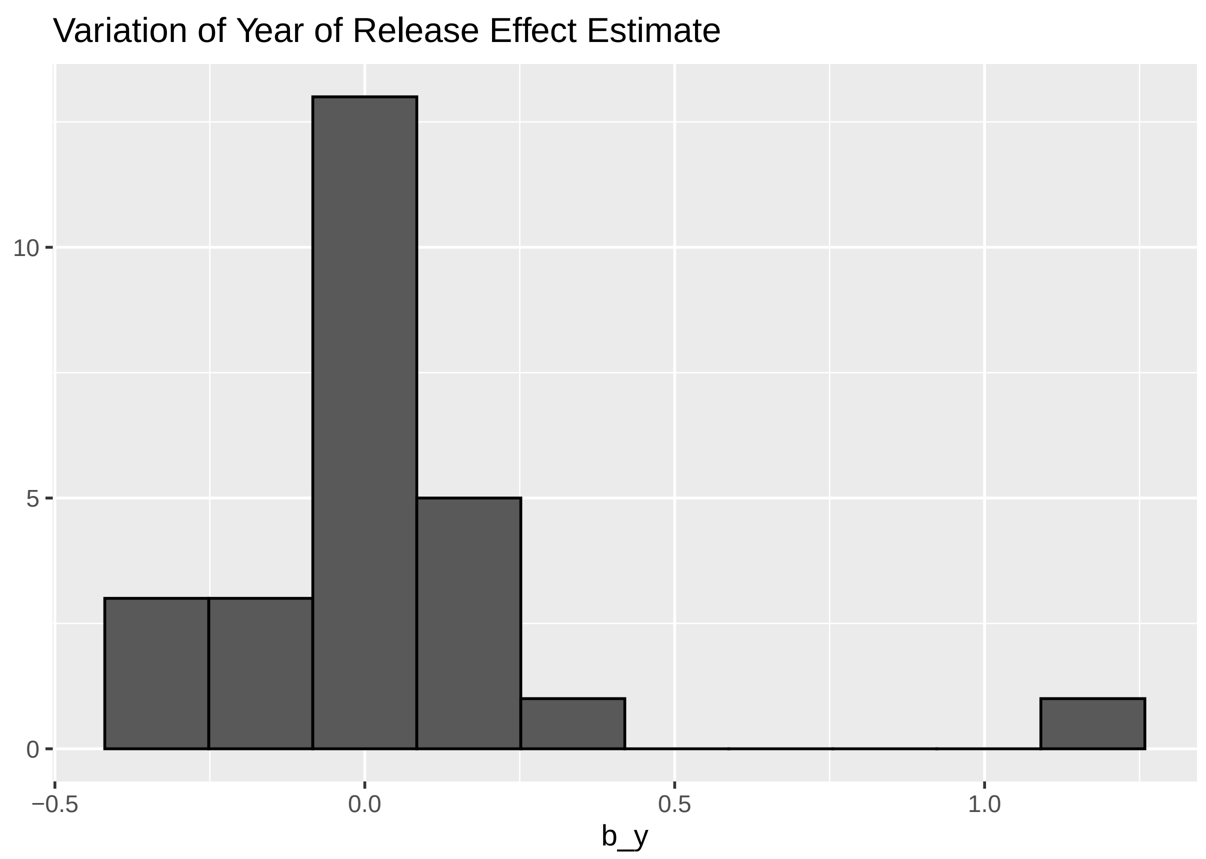
Computation of Year of Release Effect Estimate

Year of release effect is calculated as follows:

```
#Adding Year of Release Effect into Model

yr_avgs <- train_set %>%
  left_join(platform_avgs, by='Platform') %>%
  left_join(genre_avgs, by='Genre') %>%
  left_join(publisher_avgs, by='Publisher') %>%
  left_join(developer_avgs, by='Developer') %>%
  group_by(Year_of_Release) %>%
  summarize(b_y = mean(Global_Sales - mu_hat - b_p - b_g - b_pu - b_d))
```

Variation of Year of Release Effect Estimate

```
#Variation of Year of Release Effect Estimate
qplot(b_y, data = yr_avgs, bins = 10, color = I("black"))+
  ggtitle("Variation of Year of Release Effect Estimate")
```



RMSE Result

After adding year of release effect into the model, RMSE is not changed much. It might not quite helpful for modeling. Next, regularization is introduced to better model performance.

```
predicted_sales <- validation %>%
  left_join(platform_avgs, by='Platform') %>%
  left_join(genre_avgs, by='Genre') %>%
  left_join(publisher_avgs, by='Publisher') %>%
  left_join(developer_avgs, by='Developer') %>%
  left_join(yr_avgs, by='Year_of_Release') %>%
  mutate(pred = mu_hat + b_p + b_g + b_pu + b_d + b_y) %>%
  .$pred

model_5_rmse <- RMSE(predicted_sales, validation$Global_Sales)
#model_5_rmse #for result testing

rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Platform + Genre + Publisher + Developer + Yea
                                     RMSE = model_5_rmse ))

rmse_results %>% knitr::kable() #for result testing
```

Predictive Method	RMSE
Just Global Average	1.510188

Predictive Method	RMSE
Platform Effect Model	1.490115
Platform + Genre Effects Model	1.489390
Platform + Genre + Publisher Effects Model	1.409233
Platform + Genre + Publisher + Developer Effects Model	1.329279
Platform + Genre + Publisher + Developer + Year of Release Effects Model	1.326855

Section 3.5.7. Regularization in the Model

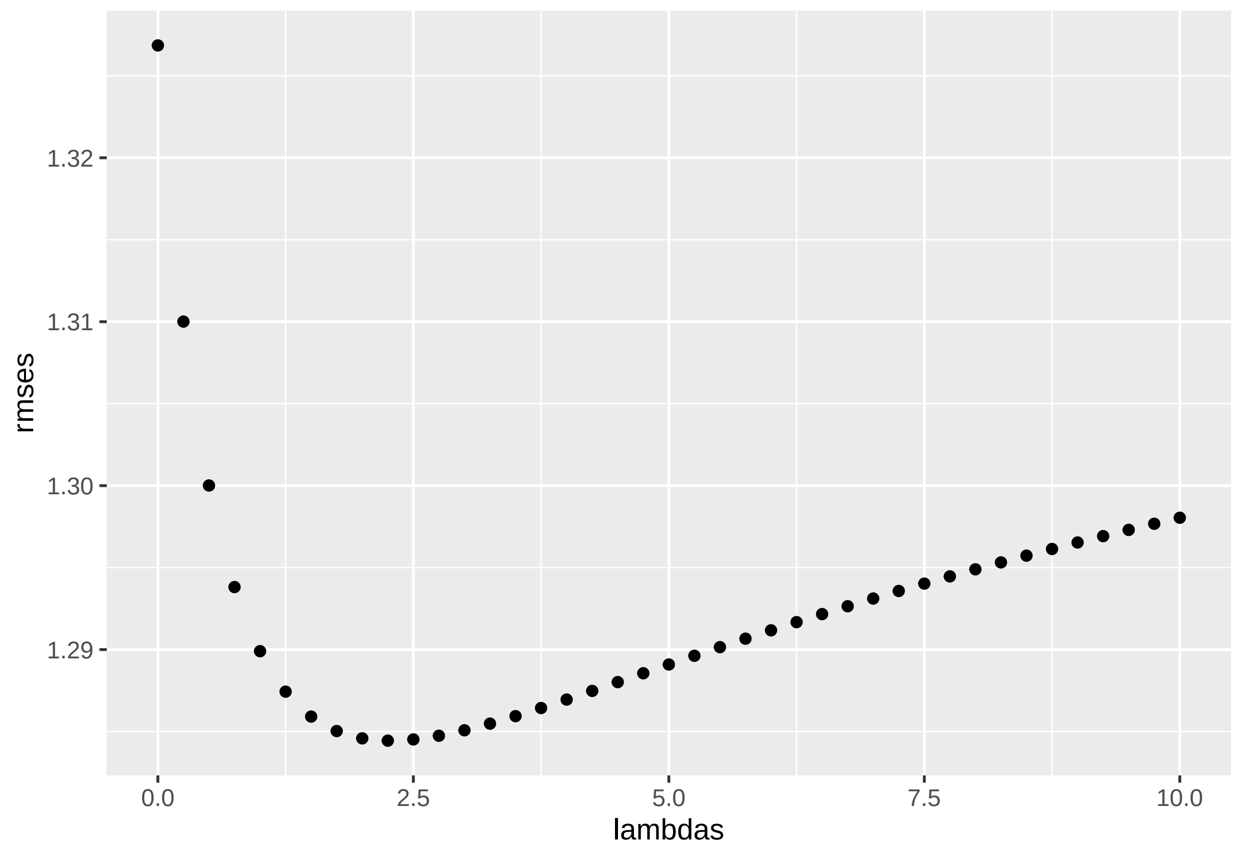
In the previous section, the improvement in RMSE was still limited. One of the reasons is the large variation between each independent variable. There might have more uncertainty on the effect and it would cause a larger estimates of effect (either positive or negative). Those estimation will probably have a large error during modeling and even increase RMSE. For this, regularization is introduced.

Regularization is a technique used to reduce the error in prediction by adding a penalty into the model to improve the function fitting on the training set and avoid overfitting. It penalizes large estimate generated by using the small dataset and control the total variability of the effects. When the effect is large, the penalty will also be increased.

Before computing the regularized estimate of effects, a penalty tuning parameter, so called lambda is required. In this project, cross validation is used on the training set to select the optimal lambda, which minimizes RMSE the most. From information below, the optimal lambda for all effects is 2.25.

Section 3.5.7.1 Choosing Penalty Term

```r
#Regularization

#Choosing the penalty terms using cross-validation
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu_hat <- mean(train_set$Global_Sales)

  platform_reg_avgs <- train_set %>%
    group_by(Platform) %>%
    summarize(b_p = sum(Global_Sales - mu_hat)/(n()+l))

  genre_reg_avgs <- train_set %>%
    left_join(platform_reg_avgs, by='Platform') %>%
    group_by(Genre) %>%
    summarize(b_g = sum(Global_Sales - mu_hat - b_p)/(n()+l))

  publisher_reg_avgs <- train_set %>%
    left_join(platform_reg_avgs, by='Platform') %>%
    left_join(genre_reg_avgs, by='Genre') %>%
    group_by(Publisher) %>%
    summarize(b_pu = sum(Global_Sales - mu_hat - b_p - b_g)/(n()+l))

  developer_reg_avgs <- train_set %>%
    left_join(platform_reg_avgs, by='Platform') %>%
    left_join(genre_reg_avgs, by='Genre') %>%
    left_join(publisher_reg_avgs, by='Publisher') %>%
    group_by(Developer) %>%
    summarize(b_d = sum(Global_Sales - mu_hat - b_p - b_g - b_pu)/(n()+l))
  yr_reg_avgs <- train_set %>%
    left_join(platform_reg_avgs, by='Platform') %>%
    left_join(genre_reg_avgs, by='Genre') %>%
    left_join(publisher_reg_avgs, by='Publisher') %>%
    left_join(developer_reg_avgs, by='Developer') %>%
    group_by(Year_of_Release) %>%
    summarize(b_y = sum(Global_Sales - mu_hat - b_p - b_g - b_pu - b_d)/(n()+l))

  predicted_sales <- validation %>%
    left_join(platform_reg_avgs, by='Platform') %>%
    left_join(genre_reg_avgs, by='Genre') %>%
    left_join(publisher_reg_avgs, by='Publisher') %>%
    left_join(developer_reg_avgs, by='Developer') %>%
    left_join(yr_reg_avgs, by='Year_of_Release') %>%
    mutate(pred = mu_hat + b_p + b_g + b_pu + b_d + b_y) %>%
    .$pred

  return(RMSE(predicted_sales, validation$Global_Sales))
})

qplot(lambdas, rmses)
```



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

```
## [1] 2.25
```

Section 3.5.7.2 Modeling With Regularization

To see how the model is improved using regularized estimate of effects with lambda = 2.25, the following code is used.The RMSE is dropped to 1.284444.

```
#Build the model using the lambda with the minimum RMSE
lambda <- lambdas[which.min(rmses)]

platform_reg_avgs <- train_set %>%
  group_by(Platform) %>%
  summarize(b_p2 = sum(Global_Sales - mu_hat)/(n()+lambda), n_i = n())

genre_reg_avgs <- train_set %>%
  left_join(platform_reg_avgs, by='Platform') %>%
  group_by(Genre) %>%
  summarize(b_g2 = sum(Global_Sales - mu_hat - b_p2)/(n()+lambda), n_i = n())

publisher_reg_avgs <- train_set %>%
  left_join(platform_reg_avgs, by='Platform') %>%
  left_join(genre_reg_avgs, by='Genre') %>%
  group_by(Publisher) %>%
  summarize(b_pu2 = sum(Global_Sales - mu_hat - b_p2 - b_g2)/(n()+lambda), n_i = n())

developer_reg_avgs <- train_set %>%
  left_join(platform_reg_avgs, by='Platform') %>%
  left_join(genre_reg_avgs, by='Genre') %>%
  left_join(publisher_reg_avgs, by='Publisher') %>%
  group_by(Developer) %>%
  summarize(b_d2 = sum(Global_Sales - mu_hat - b_p2 - b_g2 - b_pu2)/(n()+lambda), n_i = n())

yr_reg_avgs <- train_set %>%
  left_join(platform_reg_avgs, by='Platform') %>%
  left_join(genre_reg_avgs, by='Genre') %>%
  left_join(publisher_reg_avgs, by='Publisher') %>%
  left_join(developer_reg_avgs, by='Developer') %>%
  group_by(Year_of_Release) %>%
  summarize(b_y2 = sum(Global_Sales - mu_hat - b_p2 - b_g2 - b_pu2 - b_d2)/(n()+lambda), n_i = n())

predicted_sales <- validation %>%
  left_join(platform_reg_avgs, by='Platform') %>%
  left_join(genre_reg_avgs, by='Genre') %>%
  left_join(publisher_reg_avgs, by='Publisher') %>%
  left_join(developer_reg_avgs, by='Developer') %>%
  left_join(yr_reg_avgs, by='Year_of_Release') %>%
  mutate(pred = mu_hat + b_p2 + b_g2 + b_pu2 + b_d2 + b_y2) %>%
  .$pred

model_6_rmse <- RMSE(predicted_sales, validation$Global_Sales)
model_6_rmse #for result testing
```

```
## [1] 1.284444
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame("Predictive Method"="Regularized Effects (Platform + Genre + Publi
                                     RMSE = model_6_rmse ))

rmse_results %>% knitr::kable() #for result testing
```

Predictive Method	RMSE
Just Global Average	1.510188
Platform Effect Model	1.490115
Platform + Genre Effects Model	1.489390
Platform + Genre + Publisher Effects Model	1.409233
Platform + Genre + Publisher + Developer Effects Model	1.329279
Platform + Genre + Publisher + Developer + Year of Release Effects Model	1.326855
Regularized Effects (Platform + Genre + Publisher + Developer + Year of Release) Model	1.284444

Section 4. Results

After regularizing the effects, the RMSE is decreased from 1.510188 to 1.284444. This indicates that the model can predict better than using the global rating average.

Section 5. Conclusiong

Section 5.1. Summary

Sales forecasting is vital for business company. It can help the firm to make better business planning, budgeting and predict the short-term and long-term performance in term of profit.

In this project, it allowed us to understand the video game sales dataset (which is open-source and can be found on Kaggle), perform data cleaning and learn how to build a sale prediction model. The model took account of global average, the similarity within each selected independent variables. The variables are *Platform*, *Genre*, *Publisher*, *Developer* and *Year of Release* to build a . Regularization was then used for all the above independent variables to enhance the system. The RMSE is largely dropped from 1.510188 to 1.284444. It indicates that the video game sales is affected by the above variables and after those variables are included in the model, it is possible to have a fairly reliable sales prediction model with the performance better than simply taking the average of the entire dataset.

Although, the RMSE is decreased, it does not imply that the model can forecast accurate video game sales. It might be the case that this model did well in this small dataset but the ability of prediction might be worse in another dataset. Hence, the following sections will investigate the limitation in this project and the future work can be done to improve the model prediction.

Section 5.2. Limitation

Section 5.2.1. Limited Dataset

As mentioned earlier, there are a lot of missing values in the dataset, especially quantitative variables (for example, *Critic_score*, *User_score* and *Rating*) so only less than 10,000 observations can be used in modeling. In order to have more accurate model, it is better to have a larger dataset with less missing values.

Section 5.2.2. Lack of Sales History for New Video Game

Sales prediction is based on how profitable each video game was able to achieve in the past. For a new video game, it might not have much revenue history to be tracked and those video games can be popular suddenly. Above issues can cause inaccurate sales forecasting.

Section 5.2.3. Time-Critical Modeling

The dataset used in this project is scrapped from website in December 2016. The sales information might be not the latest version since the sales change over the times. In order to perform sales prediction precisely, the modeling requires to be completed with the latest dataset.

Section 5.3. Future work

Section 5.3.1. Cross-Validation

In this project, the dataset is only split into training and validation datasets once to build the sales prediction model. In fact, the data set can be repeatedly split into the above two datasets, so called cross-validation, so as to utilize all significant data for training and get better and more stable prediction.

Section 5.3.2. Additional Variables

Video game business is quite seasonal and some of them might make profit during holidays such as New Year and Christmas. Moreover, some firm might increase the demand and visibility of video game by sales promotion within certain weeks during the year. Therefore, apart from the video game's basic information, model can be tuned by including seasonal information and holiday effects on the video game to improve model performance.

Section 5.3.3. Real Time Capability

As mentioned above (*Section 5.2.3.*), sales prediction requires to be time-critical so that the company can gain prompt insight, update the forecast based on the change in demand/market and make the business decision quickly. Thus, it is suggested to produce the model to re-forecast in real time and fine-tune the model over time to improve the accuracy and create more refined future forecasts.