Capstone Project - Video Games Sales Prediction

Fidan Gasim

1/3/2021

Contents

Preface

This report was prepared as part of the capstone project for HarvardX's Data Science Professional Certificate Program[1].

1. Introduction

1.1 Overview

2020 was dominated by the coronavirus pandemic and the resulting worldwide lockdowns. While many industries and businesses suffered during this time, one industry in particular saw record-breaking sales. As more consumers stayed home, video games filled a void for those looking for entertainment. In fact, it is estimated that COVID-19 lockdowns resulted in a 20% increase in gaming sales to nearly $180 billion in 2020[2]. Experts predict that this number will only keep growing, as new generation consoles like the Playstation 5 hit the market in 2021[3].

Although video games have been around since the 1950s, their popularity over the last couple of decades has skyrocketed. This can be attributed in large part to the rise of TV ownership in households as well as the introduction of gaming consoles and home computers. With the proliferation of mobile phones and high speed internet, video games' sales soared even higher.

2020 saw the gaming industry's sales overtake those of the movies and North American sports industries combined[4]. While the pandemic has contributed to many more new users adopting video games this year, it is interesting to look at what other factors contribute to a video game's success. Why do some video games become blockbusters while others only have a small number of users? How can we predict how successful a game will be in terms of units sold? As more companies enter the video game domain, it is invaluable to gain an understanding of what features have the greatest impact on sales' success.

For the purposes of this project, I will be building a model to predict global sales of video games based on the features available in the dataset. I will be using methods and tools that have been learned during HarvardX's Data Science Professional Certificate program[5] as well as concepts studied outside the course.

This document is structured as follows: Section 1 describes the videogames dataset, outlines the goals of the project and provides a summary of the key steps taken to achieve them. Section 2 outlines the data visualization techniques used to understand the dataset, any insights gained, and the data cleaning and preparation process. Section 3 explains the modeling methods used to tackle the machine learning challenge. Section 4 presents the modeling results and an evaluation of the models' performances. Section 5 provides a summary of the project, its limitations, and any future work that can be conducted.

1.2 Video Games Dataset

The Video Game Sales with Ratings[6] dataset I will be using consists of sales information for over 15,000 games that have sold over 100,000 copies, along with other descriptive features such as platform, genre, year of release, publisher, rating, critic and user scores, etc. The dataset by Rush Kirubi[7] combines data from

[1]https://www.edx.org/professional-certificate/harvardx-data-science
[2]https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990
[3]https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990
[4]https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990
[5]https://www.edx.org/professional-certificate/harvardx-data-science
[6]https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings
[7]https://www.kaggle.com/rush4ratio

another video games dataset[8] with a web scrape from Metacritic[9]. The initial video games sales dataset by Gregory Smith[10] was put together using a web scrape from VGChartz[11], a video game sales tracking website.

The initial set of features in the dataset were Name, Platform, Year of Release, Genre, Publisher, NA Sales, EU Sales, JP Sales, Other Sales, GlobalSales. The additional features, added later by Rush Kirubi, were Critic Score, Critic Count, User Score, User Count, Developer, and Rating.

Acocrding to Kirubi, there are missing observations as Metacritic only covers a subset of the platforms. Also, some games do not have all the observations of the additional variables. Complete cases (with no NAs) are approximately 6,900.

1.3 Model Evaluation

When working on any data science project, it is important to evaluate the performance of the machine learning model and how it generalizes to new, unseen data. The model evaluation involves comparing the predicted values with the actual outcomes. We need to examine whether the model actually works and how well we can trust its predictions[12].

Evaluating a model's performance involves using a hold-out test set (data not seen by the model), which provides an "unbiased estimate of learning performance"[13]. Typically, data that is used to build the model is not used to evaluate it. Using the training set for both building and testing the algorithm can lead to overfitting, which is when the model remembers the whole training set and does not adapt well to new data.

While there are many types of metrics, such as Classification Accuracy, Confusion Matrix, Area Under Curve (AUC) and F-Measure, that are used to evaluate machine learning algorithms, for this project we will be using a loss function, specifically RMSE (Root Mean Squared Error), to evaluate the performance of our models.

Root Mean Squared Error is the square root of the average squared distance between the actual outcomes and the predicted values. In other words, it measures the difference between predicted values and actual outcomes.

The RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

where $y_{u,i}$ is the observed value for observation i and $\hat{y}_{u,i}$ is the predicted value.

1.4 Process Outline

Besides data collection, the main steps in a machine learning project include:

1. Project Understanding: understanding the project's goals and creating a workflow of key steps.
2. Data Preparation: downloading, importing and preparing the dataset for analysis.
3. Data Exploration: gaining insights from the data and explore any assumptions before modeling. Any necessary data transformations as well as key features/predictors would be identified in this step.

[8]https://www.kaggle.com/gregorut/videogamesales
[9]https://www.metacritic.com/
[10]https://www.kaggle.com/gregorut
[11]https://www.vgchartz.com/
[12]https://heartbeat.fritz.ai/introduction-to-machine-learning-model-evaluation-fa859e1b2d7f
[13]https://heartbeat.fritz.ai/introduction-to-machine-learning-model-evaluation-fa859e1b2d7f

4. Data Preprocessing: cleaning and transforming the data, such as feature selection, scaling, removing unnecessary information, etc.
5. Modeling Methods: researching and selecting modeling approaches that work best for the type of dataset.
6. Data Modeling: Creating, training and testing the selected models, including any fine tuning of parameters.
7. Model Evaluation: Evaluating the results and model's performance.
8. Communication: Presenting the final results along with any limitations or recommendations for future work.

We start off this project by downloading and importing the dataset from kaggle. After a few preprocessing measures, such as formatting variables, we split the dataset into two subsets: vgTrain and vgTest. The vgTrain dataset is used to explore the data, select features, and train the machine learning models, while vgTest is used to test the final algorithm.

RMSE (Root Mean Squared Error) will be used to evaluate how close our predictions are to the true values in the test set. The test set is the final hold-out test set and is not used for training, developing or selecting the machine learning algorithm. We predict video game sales in the test set as if they were unknown.

In the following section, we will analyze the data to look for trends and any relationships between the features and outcomes. We will produce charts, tables, and summary statistics to show these relationships and trends. The insights gained during the data exploration process will help us build our machine learning model.

In order to build an effective model, we need to identify the most important features that will help predict global sales with the highest accuracy. Since our outcome variable (Global Sales) is continuous in nature, we will start by training a simple linear regression model, followed by three other models: Elastic Net, Support Vector Machine, and Random Forest.

The Random Forest model produces the lowest validation error (RMSE) during the training phase and is therefore selected to make predictions using the test set. This gives us an "independent final check on the accuracy of the best model"[14].

2 Data Exploration and Preprocessing

In this section, we start off by installing the necessary packages and loading the corresponding libraries which we'll be using to analyze our data and train our models.

```r
# -> Load libraries ####
library(tidyverse)
library(readr)
library(tidyr)
library(dplyr)
library(ggplot2)
library(caret)
library(data.table)
library(knitr)
library(kableExtra)
library(lubridate)
library(Matrix.utils)
library(DT)
library(RColorBrewer)
library(ggthemes)
```

[14] https://machinelearningmastery.com/machine-learning-in-r-step-by-step/

```r
library(scales)
library(purrr)
library(devtools)
library(ggrepel)
library(splitstackshape)
library(tree)
library(gam)
library(gridExtra)
library(matrixStats)
library(GGally)
library(corrplot)
library(lares)
library(polycor)
library(lsr)
library(Boruta)
library(e1071)
library(randomForest)
library(glmnet)
library(ade4)
library(FSelector)
library(naniar)
library(klaR)
library(kernlab)
library(tinytex)
```

Because the dataset can't be downloaded directly onto RStudio using the Kaggle link, it is uploaded to my GitHub account. It is then downloaded and imported using the GitHub URL link where it is stored.

First, we'll explore the basics of the dataset.

```r
class(videogames) # type of dataset
```

```
## [1] "data.frame"
```

```r
dim(videogames) # no. of rows and columns in the dataset
```

```
## [1] 16719    16
```

The videogames dataset is of class type "data frame".

There are 16,719 observations and 16 features (columns). The data types of each column are as follows:

```r
str(videogames) # structure of the dataset - data types of each column
```

```
## '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: int  2006 1985 2008 2009 1996 1989 2006 2006 2009 1984 ...
##  $ 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" NA "8.3" "8" ...
##  $ User_Count     : int  322 NA 709 192 NA NA 431 129 594 NA ...
##  $ Developer      : chr  "Nintendo" NA "Nintendo" "Nintendo" ...
##  $ Rating         : chr  "E" NA "E" "E" ...
```

```r
head(videogames) # first few rows of the dataset
```

Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_S
Wii Sports	Wii	2006	Sports	Nintendo	41.36	28.96	
Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	
Mario Kart Wii	Wii	2008	Racing	Nintendo	15.68	12.76	
Wii Sports Resort	Wii	2009	Sports	Nintendo	15.61	10.93	
Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	1
Tetris	GB	1989	Puzzle	Nintendo	23.20	2.26	

The features in the dataset are as follows:

- Name – Name of the video game
- Platform - Platform to which the game was released
- Year - Year of the game's release
- Genre - Genre of the game
- Publisher - Publisher of the game
- NA Sales - Sales in North America (in millions of units)
- EU Sales - Sales in Europe (in millions of units)
- JP Sales - Sales in Japan (in millions of units)
- Other Sales - Sales in the rest of the world (in millions of units)
- Global Sales - Total worldwide sales (in millions of units)
- Critic Score - Aggregate score by Metacritic staff
- Critic Count - The number of critics to come up with the Critic score
- User Score – Aggregate score by Metacritic's subscribers
- User Count - Number of users who gave the User Score
- Developer - Party responsible for creating the game
- Rating - The ESRB ratings – age limits for games

We can see that most of the character variables are actually categorical in nature. We will convert these into factors later.

```r
summary(videogames) # summary statistics of the dataset
```

```
##      Name             Platform         Year_of_Release      Genre          
##  Length:16719       Length:16719       Min.   :1980.000   Length:16719      
##  Class :character   Class :character   1st Qu.:2003.000   Class :character  
##  Mode  :character   Mode  :character   Median :2007.000   Mode  :character  
##                                        Mean   :2006.487                     
##                                        3rd Qu.:2010.000                     
##                                        Max.   :2020.000                     
##                                        NA's   :269                          
##   Publisher            NA_Sales           EU_Sales       
##  Length:16719       Min.   : 0.0000000   Min.   : 0.0000000
##  Class :character   1st Qu.: 0.0000000   1st Qu.: 0.0000000
##  Mode  :character   Median : 0.0800000   Median : 0.0200000
##                     Mean   : 0.2633303   Mean   : 0.1450248
##                     3rd Qu.: 0.2400000   3rd Qu.: 0.1100000
##                     Max.   :41.3600000   Max.   :28.9600000
##
##     JP_Sales           Other_Sales          Global_Sales
##  Min.   : 0.00000000   Min.   : 0.00000000   Min.   : 0.0100000
##  1st Qu.: 0.00000000   1st Qu.: 0.00000000   1st Qu.: 0.0600000
##  Median : 0.00000000   Median : 0.01000000   Median : 0.1700000
##  Mean   : 0.07760213   Mean   : 0.04733178   Mean   : 0.5335427
##  3rd Qu.: 0.04000000   3rd Qu.: 0.03000000   3rd Qu.: 0.4700000
##  Max.   :10.22000000   Max.   :10.57000000   Max.   :82.5300000
##
##   Critic_Score      Critic_Count       User_Score         User_Count
##  Min.   :13.00000   Min.   :  3.00000   Length:16719       Min.   :    4.0000
##  1st Qu.:60.00000   1st Qu.: 12.00000   Class :character   1st Qu.:   10.0000
##  Median :71.00000   Median : 21.00000   Mode  :character   Median :   24.0000
##  Mean   :68.96768   Mean   : 26.36082                      Mean   :  162.2299
##  3rd Qu.:79.00000   3rd Qu.: 36.00000                      3rd Qu.:   81.0000
##  Max.   :98.00000   Max.   :113.00000                      Max.   :10665.0000
##  NA's   :8582       NA's   :8582                           NA's   :9129
##   Developer            Rating
##  Length:16719       Length:16719
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##
```

By looking at the summary statistics of the dataset, we can see that there are missing values in some of the columns. We will deal with these later. We can also see that the numeric variables are on different scales, which we will have to adjust for some of our models.

2.1 Data Transformation

Before splitting the dataset into training and test sets, we will make a few basic changes to the variables. This won't be considered data snooping/leaking as we're not modifying any of the variables based on information that could be found only in the test set. Other changes, like dealing with missing values, scaling, feature selection, etc., will be done at a later stage.

- Remove regional sales columns we are not interested in using. We will focus only on Global Sales which is an aggregate of all the regional sales columns.
- Rename columns for ease of use (eg. 'Year of Release').
- Create new column that combines the platforms by manufacturer. This could be interesting to look at for visualization purposes as it aggregates all the information for each vendor. Perhaps it could also be more relevant for modeling since the dataset only covers years until 2016, thus excluding any consoles/platforms that were released since.
- Convert non-numeric values to numeric – some numeric columns were loaded as character variables.
- Rescale the User Score column to match the Critic Score column for visualization purposes.
- Convert categorical variables to factors - works better for statistical modeling

The following code covers all these steps:

```r
# Remove regional sales columns (we are only interested in Global Sales)
videogames<-videogames%>%dplyr::select(-c("EU_Sales","JP_Sales","NA_Sales","Other_Sales"))

# Rename columns for ease of use
videogames<-videogames%>%
  rename("Year" = Year_of_Release)

# Create new column for the age of the game (number of years since release)
videogames<-videogames%>%mutate(Age=2020-(Year))

# Create new column - Combine platforms by manufacturer (to reduce number of categories)
nintendo<-c("3DS","DS","GB","GBA","N64","GC", "NES","SNES","Wii","WiiU")
sony<-c("PS","PS2","PSP","PS3","PS4","PSV")
sega<-c("GEN","SCD","DC","GG")
microsoft<-c("XB","X360", "XOne")
other<-c("2006","3DO","NG","PCFX","TG16")

Platform_Type<-function(x){
  if (x %in% sony == TRUE) {return('Sony')}
  else if(x %in% microsoft == TRUE) {return('Microsoft')}
  else if(x %in% nintendo == TRUE) {return('Nintendo')}
  else if(x %in% sega == TRUE) {return('Sega')}
  else{return('OTHER')}
}

videogames$Platform_Type<-sapply(videogames$Platform, Platform_Type)

# Convert non-numeric values to numeric
videogames$Year<-as.numeric(as.character(videogames$Year))
videogames$User_Count<-as.numeric(as.character(videogames$User_Count))
videogames$User_Score<-as.numeric(as.character(videogames$User_Score))
videogames$Critic_Count<-as.numeric(as.character(videogames$Critic_Count))
videogames$Critic_Score<-as.numeric(as.character(videogames$Critic_Score))

# Rescale the User Score column to match Critic Score column
videogames$User_Score<- as.numeric(as.character(videogames$User_Score)) *10

# Convert categorical variables to factors - works better for statistical modeling
videogames[sapply(videogames, is.character)] <-
  lapply(videogames[sapply(videogames, is.character)], as.factor)
```

2.2 Data Splitting

The dataset is randomly split into a training and test set, called vgTrain and vgTest, respectively. VgTrain accounts for 80% of the data and vgTest accounts for the remaining 20%. The vgTrain dataset will be used for data visualization, feature selection, data imputation, and to train our models. The vgTest dataset acts as the final hold-out test set (data not seen by the model) to evaluate the model's performance. The following code achieves this:

```
# Split the videogames dataset into training and test sets
# Test set will be 20% of video games dataset
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y=videogames$Global_Sales, times = 1, p = 0.2, list = FALSE)
vgTrain <- videogames[-test_index,]
vgTest <- videogames[test_index,]

rm(test_index)
nrow(vgTrain) #check number of observations
```

```
## [1] 13374
```

```
nrow(vgTest) #check number of observations
```

```
## [1] 3345
```

vgTrain has 13,374 observations and **vgTest** has 3,345 observations.

Since our dataset is not very large, we won't be doing a further split to get a validation set. Instead we will use repeated k-fold cross-validation when training our models, which will be covered later. This works better when dealing with small to medium sized datasets.

2.3 Data Cleaning

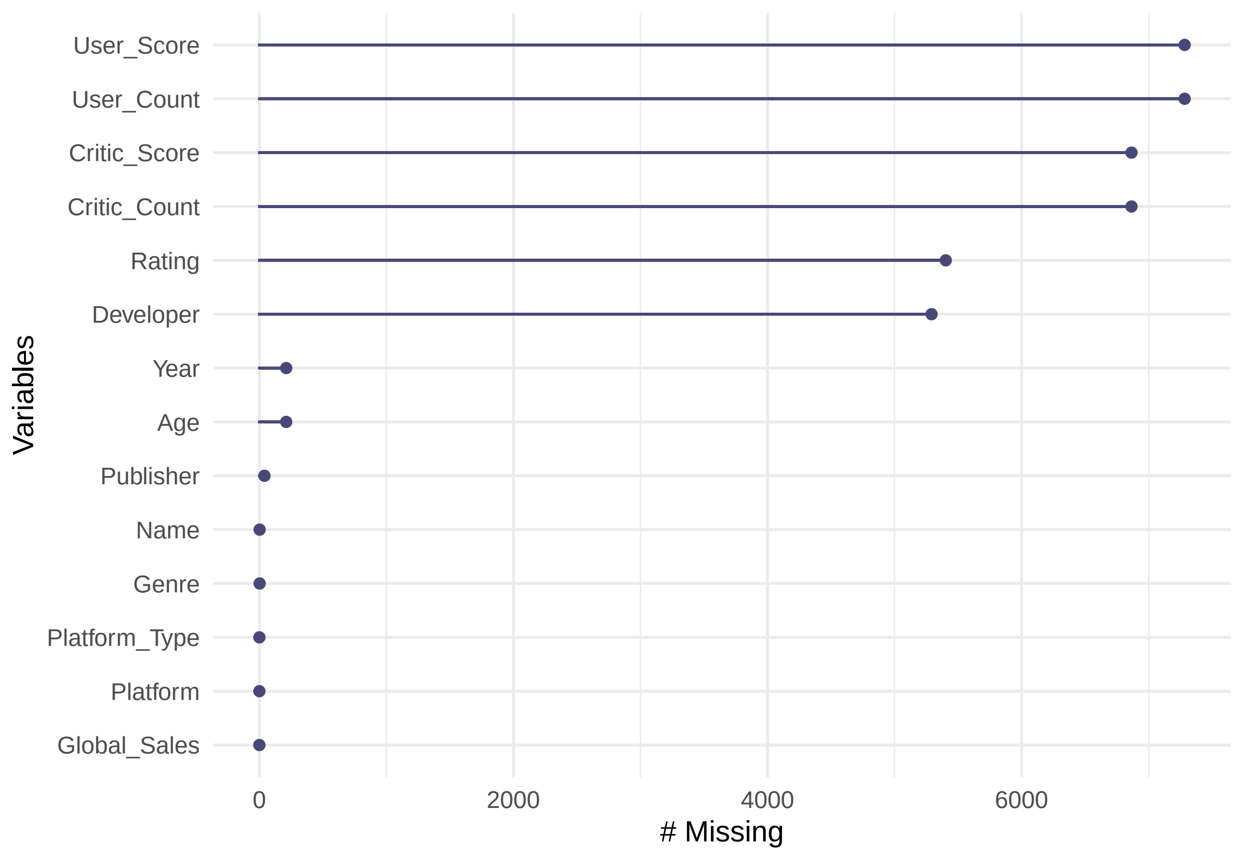
Now that we have our training set, we can work on cleaning the data. This step involves looking for missing values and deciding what actions to take to deal with them.

Number of NAs by column:

```
##           Name      Platform          Year         Genre     Publisher
##              2             0           211             2            40
##   Global_Sales  Critic_Score  Critic_Count    User_Score    User_Count
##              0          6865          6865          7283          7283
##      Developer        Rating           Age Platform_Type
##           5291          5403           211             0
```

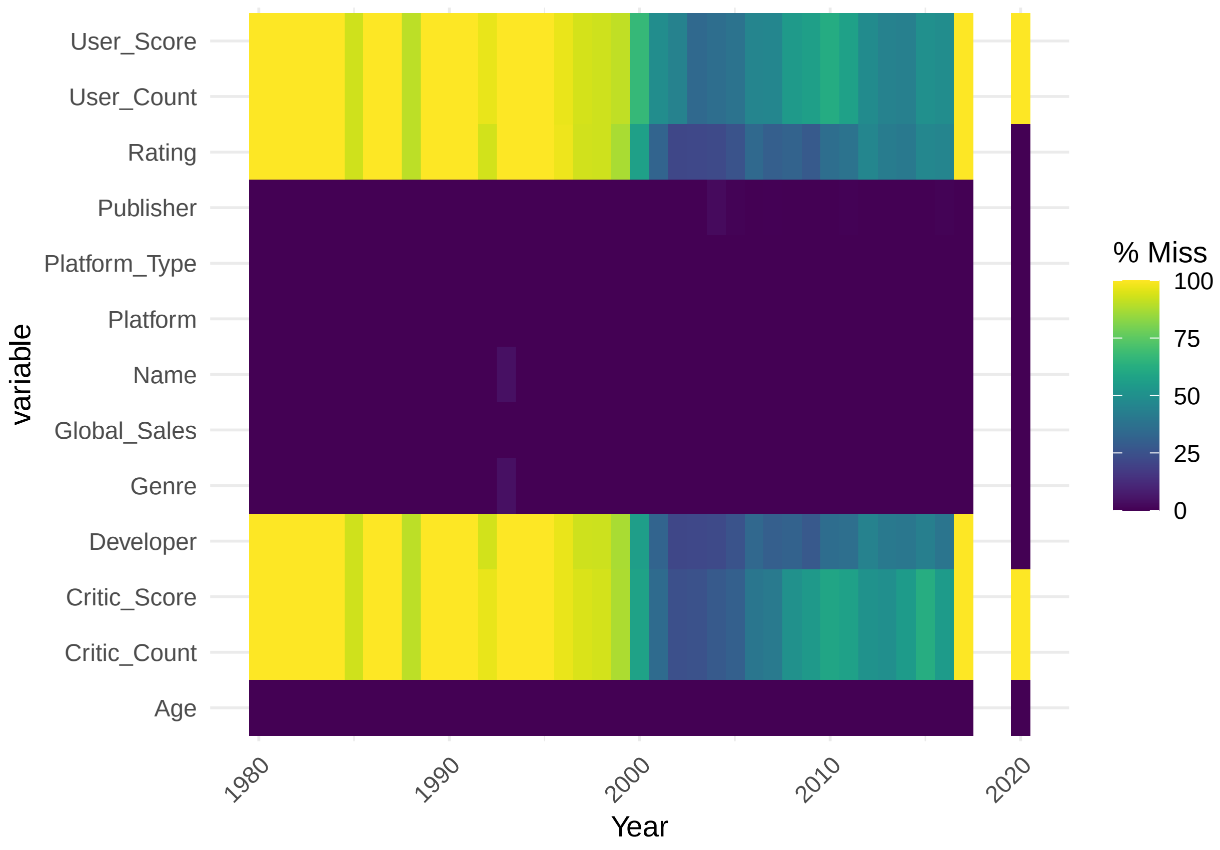
Percentage of NAs by column:

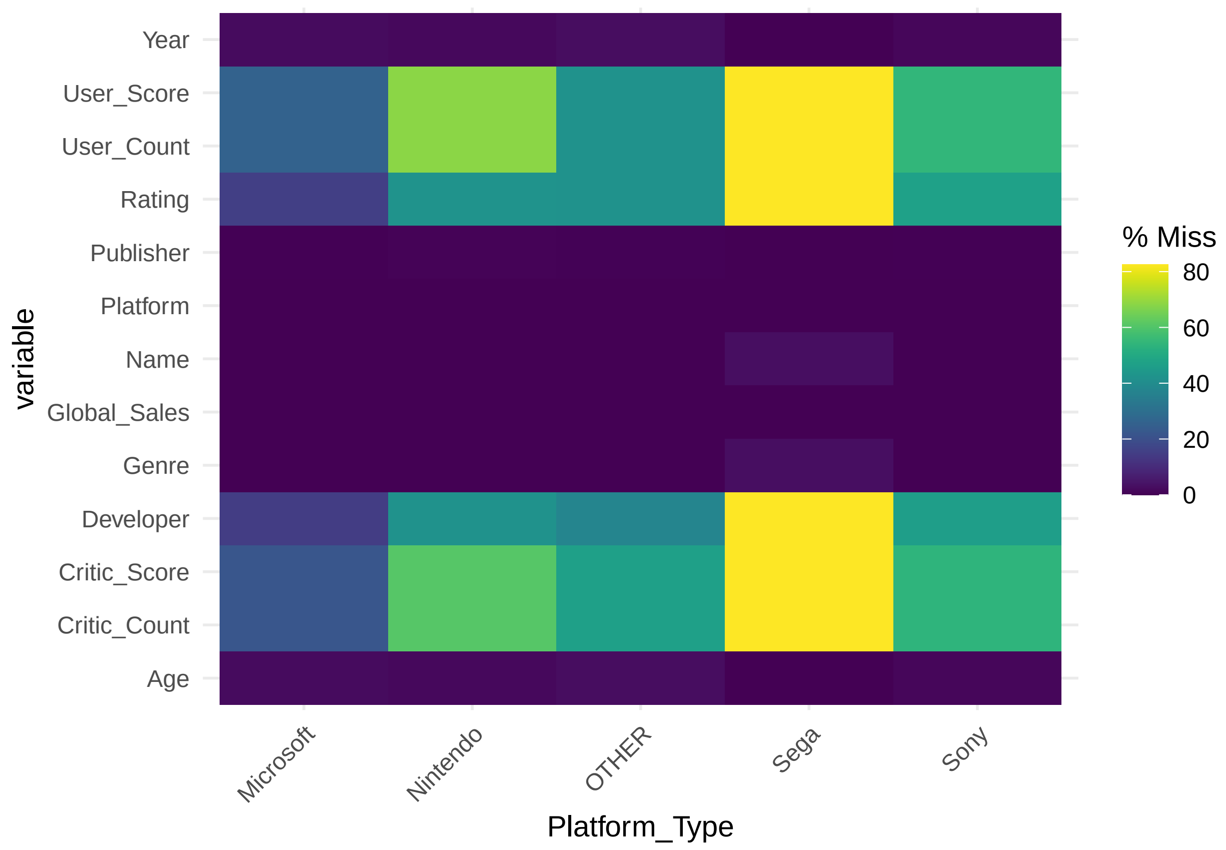
```
##             Name         Platform             Year            Genre        Publisher
## 0.0001495438911 0.0000000000000 0.0157768805144 0.0001495438911 0.0029908778226
##     Global_Sales     Critic_Score     Critic_Count       User_Score       User_Count
## 0.0000000000000 0.5133094063108 0.5133094063108 0.5445640795574 0.5445640795574
##        Developer           Rating              Age    Platform_Type
## 0.3956183639898 0.4039928218932 0.0157768805144 0.0000000000000
```



We can see that some columns such as Name, Year, Genre, and Publisher only have a small number of missing observations. These are missing at random and can be removed without having a major impact on the dataset. The other columns, however, have large chunks of data missing (in some cases over 50%). We need to make a decision about these as removing them altogether could have a severe effect on our prediction results.

As we can gather from the dataset notes by Rush Kirubi, many of these values are missing because Metacritic only covers a subset of the platforms in the dataset. We will check where the missing data lies.


User_Score
User_Count
Rating
Publisher
Platform_Type
Platform
Name
Global_Sales
Genre
Developer
Critic_Score
Critic_Count
Age
variable
1980
1990
2000
2010
2020
Year
% Miss
100
75
50
25
0



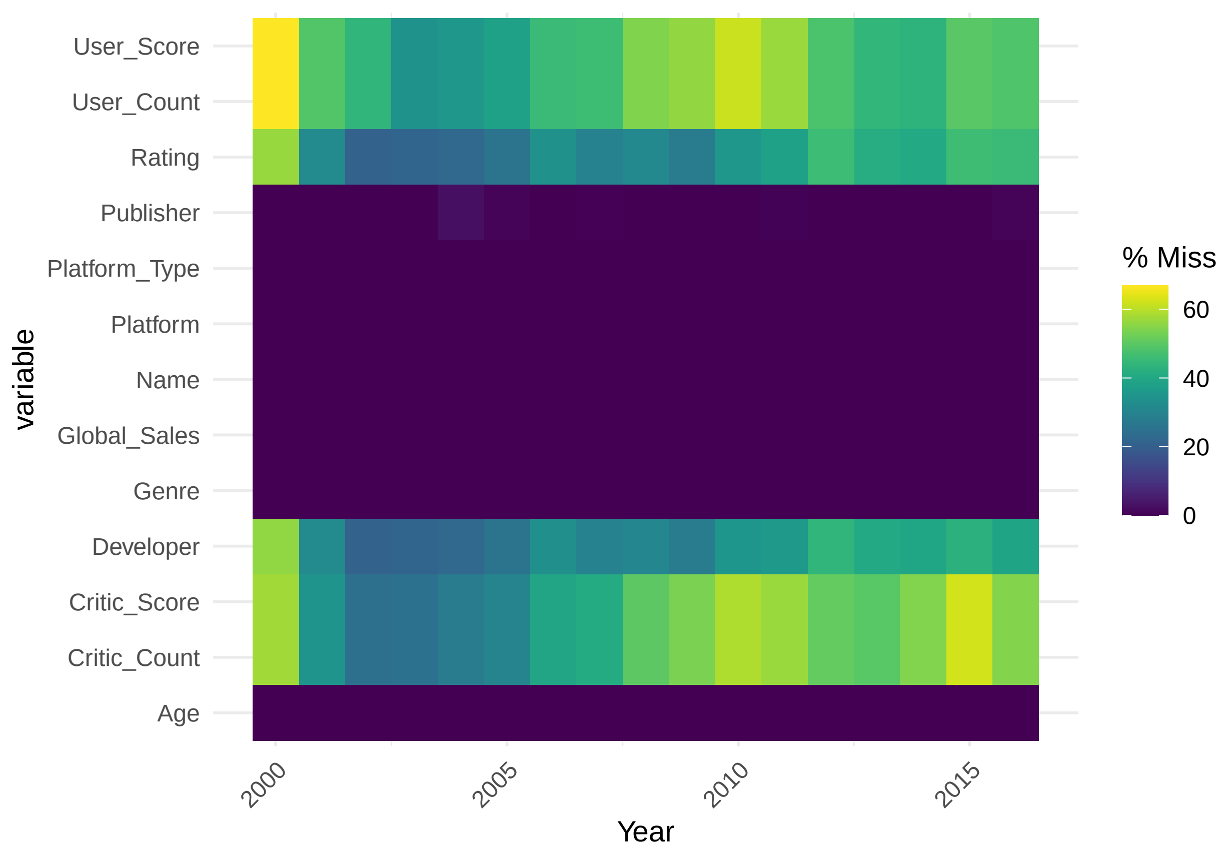
We can see that almost all the data coming from Metacritic (User Score, User Count, Critic Score, Critic Count, Rating, and Developer) is missing prior to 1999. This is because Metacritic was founded in 1999 so any data before that was not recorded for these features. We can also see missing values after 1999, which could be because some platforms are not covered by Metacritic (as mentioned by Rush Kirubi) and therefore some games never got rated.

We can also see that there are a lot of missing values for the Platform Type – Sega. This could either be because most of their games were released prior to 1999 or because Metacritic does not cover this platform.

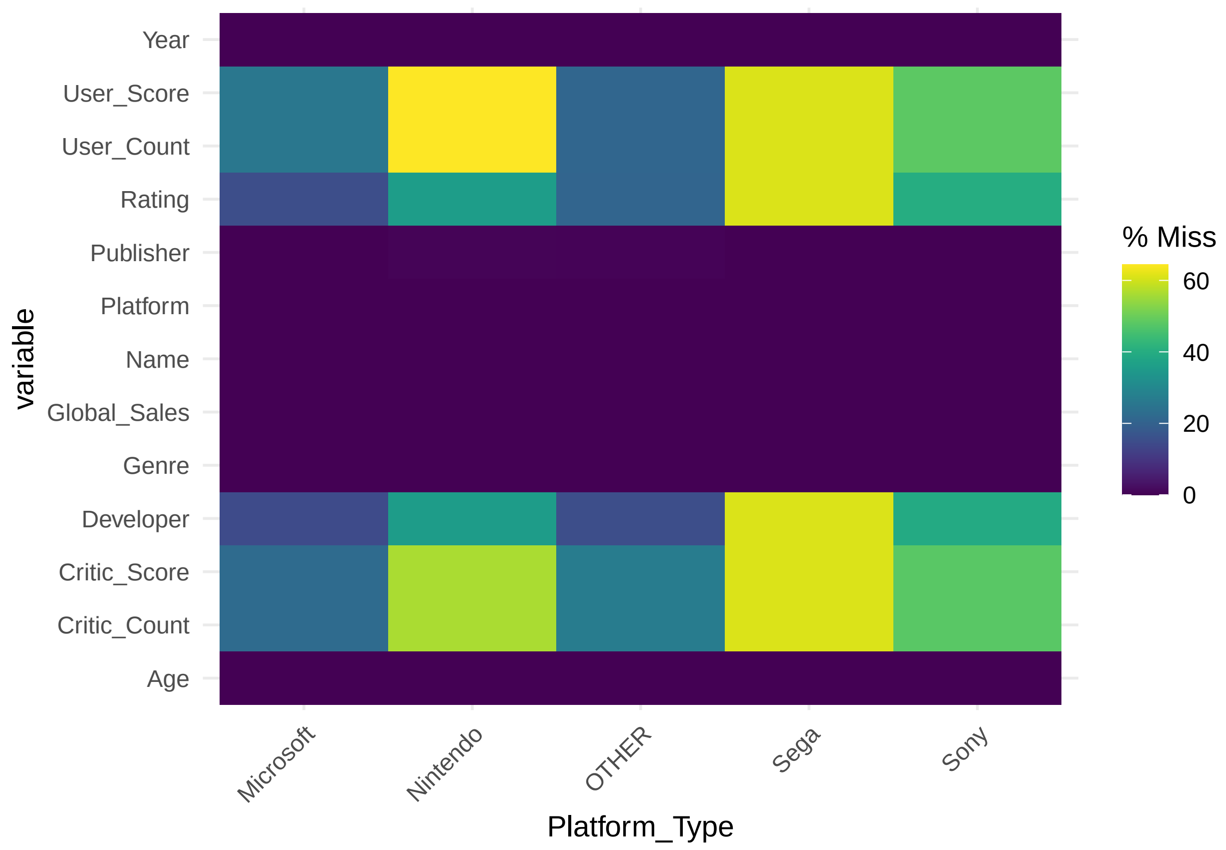
Platform_Type	Year	count
Sega	1991	1
Sega	1992	4
Sega	1993	7
Sega	1994	14
Sega	1998	6
Sega	1999	13
Sega	2000	14
Sega	2001	8
Sega	2002	1

We can see that in fact most games released by Sega were prior to 1999.

In the next step, we will remove all games released prior to 1999 (founding of Metacritic) and after 2016 (final year covered by dataset) and check whether we will have a more complete dataset to work with.



User_Score
User_Count
Rating
Publisher
Platform_Type
Platform
Name
Global_Sales
Genre
Developer
Critic_Score
Critic_Count
Age
variable
2000
2005
2010
2015
Year
% Miss
60
40
20
0



We're still left with many missing values between 1999 and 2016 across all platforms.

We will remove all observations with missing Name, Year, Genre, Publisher. This is a small percentage so it won't harm the data much. We are removing 238 observations in total. We're still left with many NAs in the features coming from Metacritic, we will deal with these later.

```
##          Name      Platform          Year         Genre     Publisher
##             0             0             0             0             0
##  Global_Sales  Critic_Score  Critic_Count    User_Score    User_Count
##             0          6752          6752          7148          7148
##     Developer        Rating           Age Platform_Type
##          5202          5307             0             0
```

We will also create a new training dataset, **vgTrain2**, with no missing values. This is helpful for data visualization purposes and also when working with certain feature selection methods that do not accept missing values.

```
##          Name      Platform          Year         Genre     Publisher
##             0             0             0             0             0
##  Global_Sales  Critic_Score  Critic_Count    User_Score    User_Count
##             0             0             0             0             0
##     Developer        Rating           Age Platform_Type
##             0             0             0             0
```

```
## [1] 5475
```

2.4 Data Visualization

In this section, we will be analyzing all the features in the training set to gain a better understanding of their nature and structure, and whether there are any significant relationships between the predictors and the target variable. This step will help us in building better models.

In some sections we will be working with the vgTrain dataset (with NAs) and in some with the vgTrain2 dataset (no NAs).

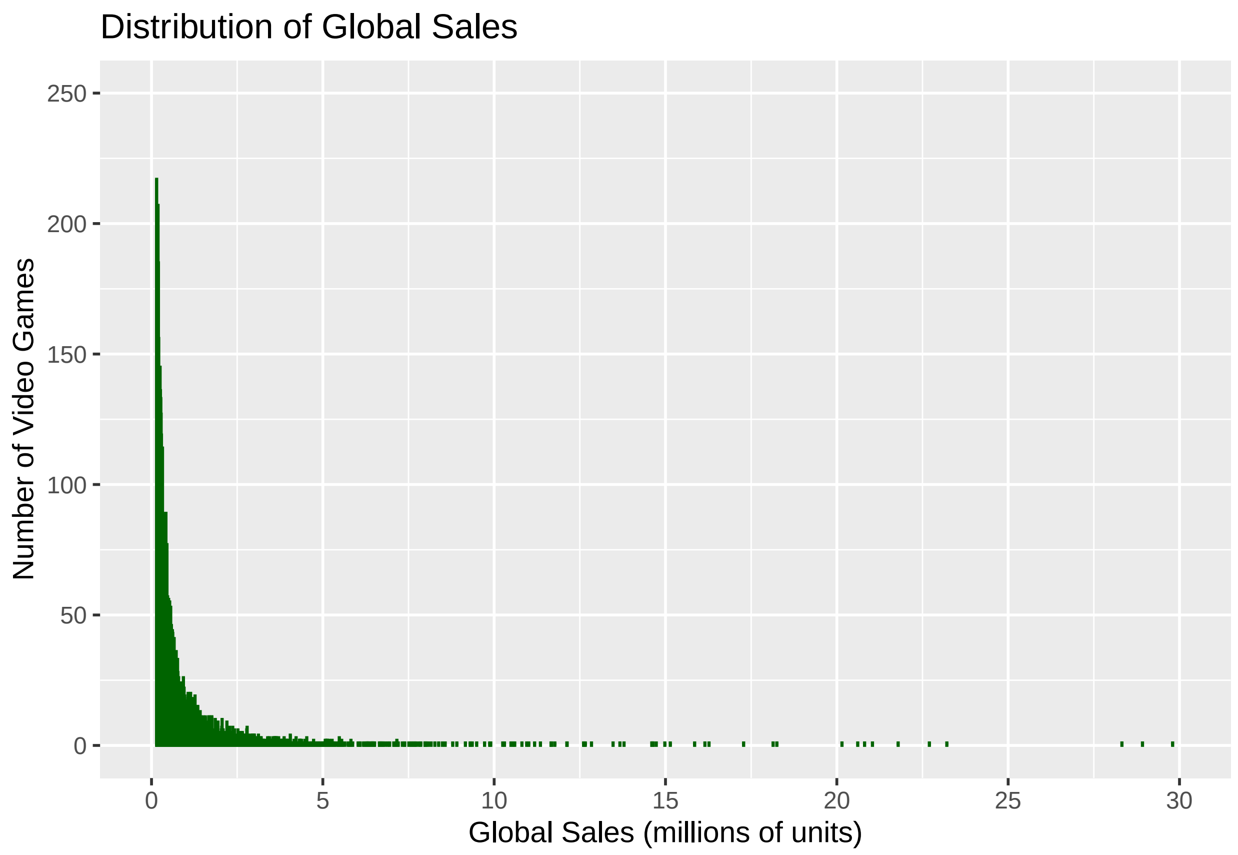
Number of Unique Video Games in the Dataset

```
## [1] 9579
```

This number is lower than the total number of observations which means some games are released more than once but on different platforms.

2.4.1 Global Sales – Target Variable

Global Sales in this dataset is recorded in millions of units rather than a currency value.



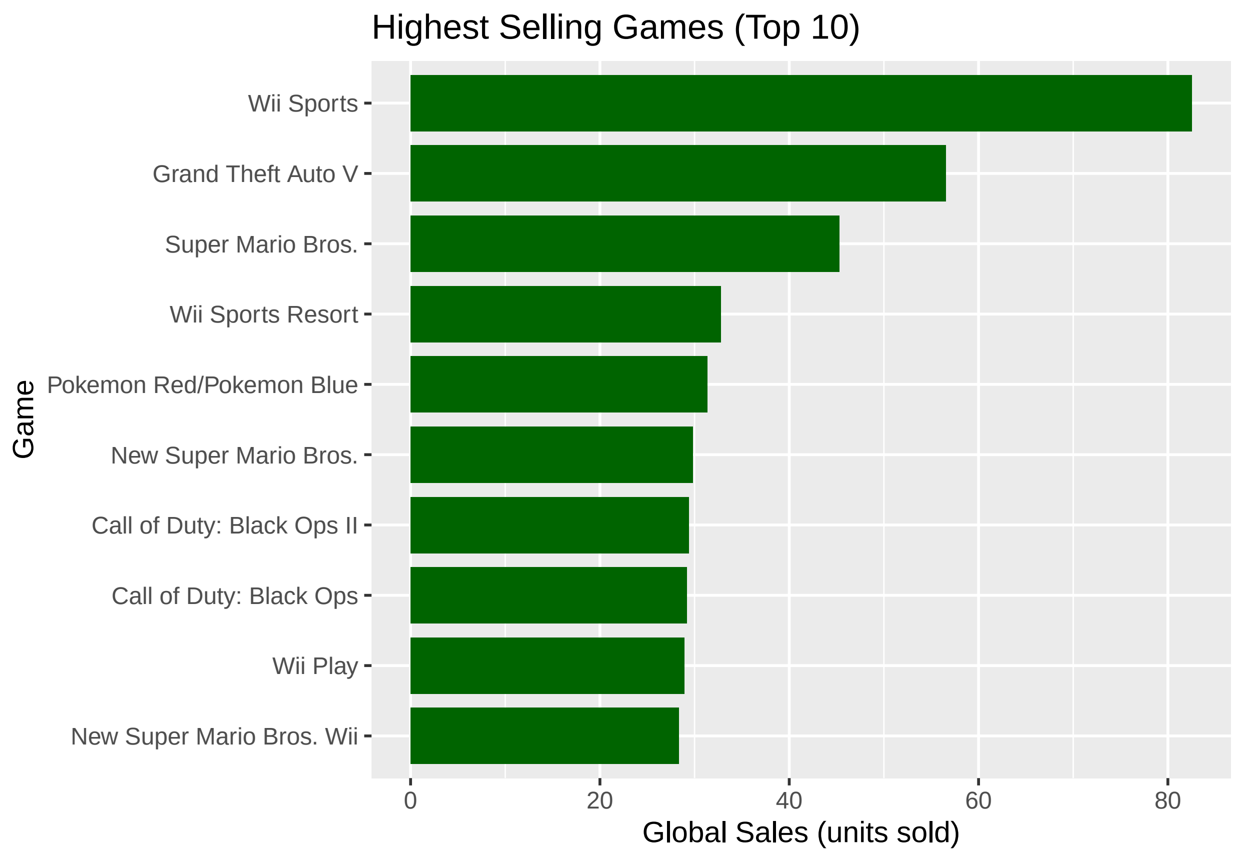
As we can see from the graph, the distribution of global sales is right skewed meaning that most video games sell less than 1 million games.

```
mean(vgTrain$Global_Sales<1.0)
```

```
## [1] 0.87477162
```

In fact, 87.5% of videogames in the dataset sold less than 1 million units, which means a very small portion of games become blockbusters.

Top 10 Highest Selling Games

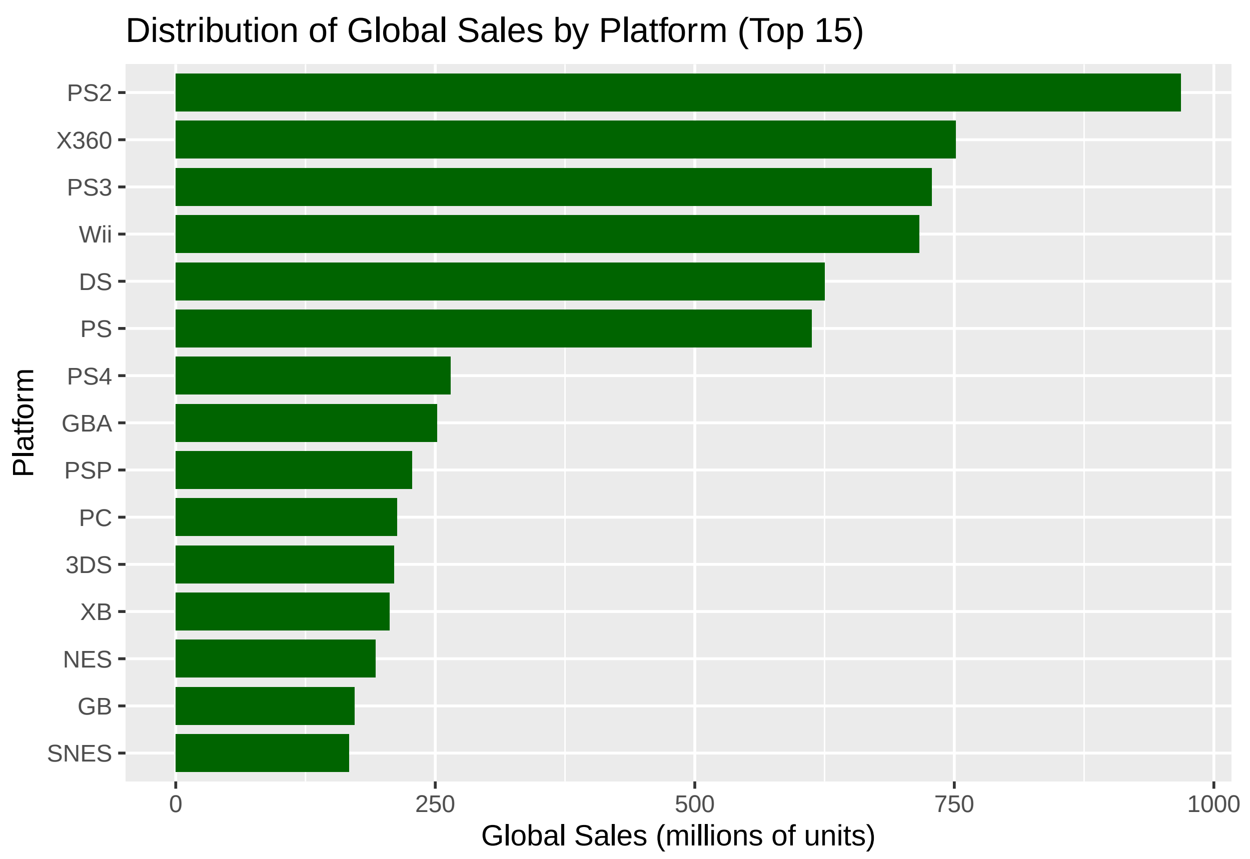


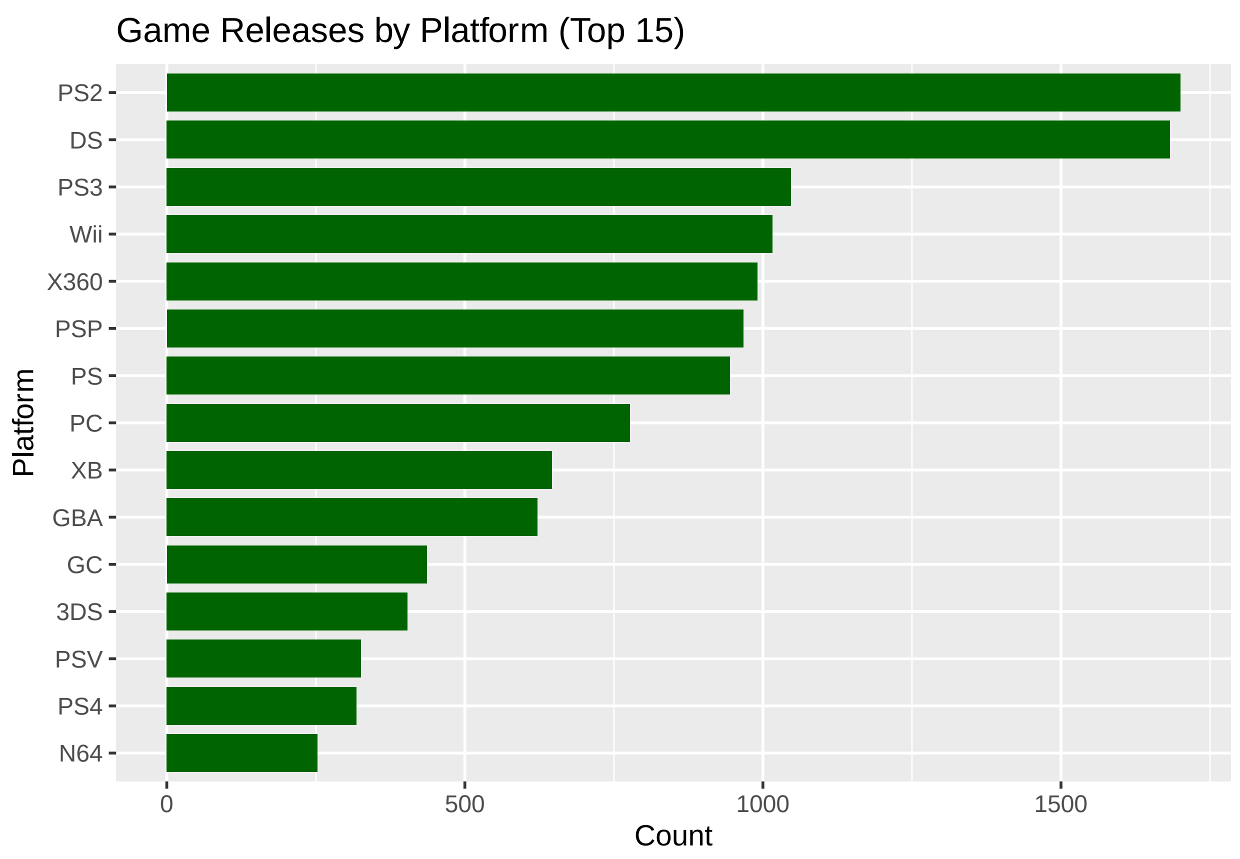
2.4.2 Platform

Number of Unique Platforms in the Dataset

```
## [1] 30
```

There are 31 unique platforms in the training dataset.

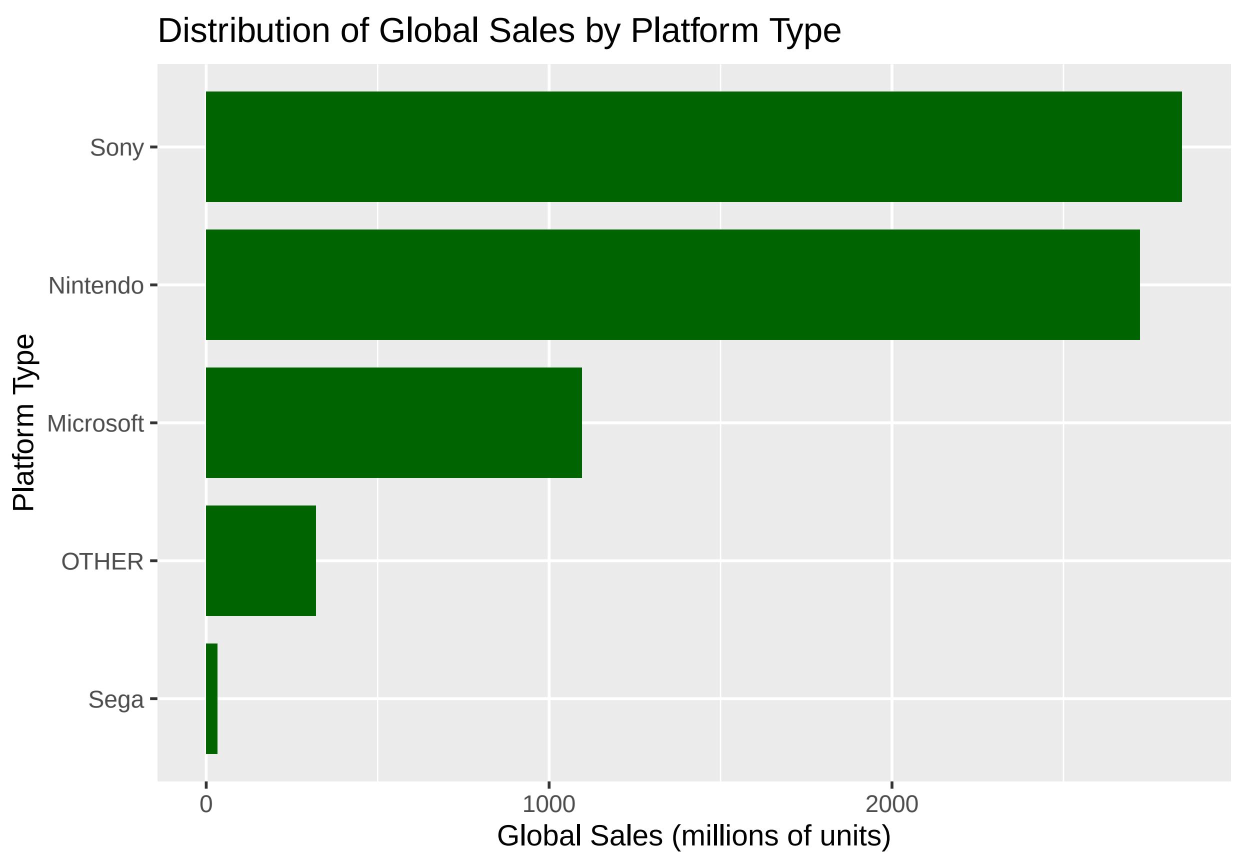

Distribution of Global Sales by Platform (Top 15)
Platform
PS2
X360
PS3
Wii
DS
PS
PS4
GBA
PSP
PC
3DS
XB
NES
GB
SNES
0
250
500
750
1000
Global Sales (millions of units)



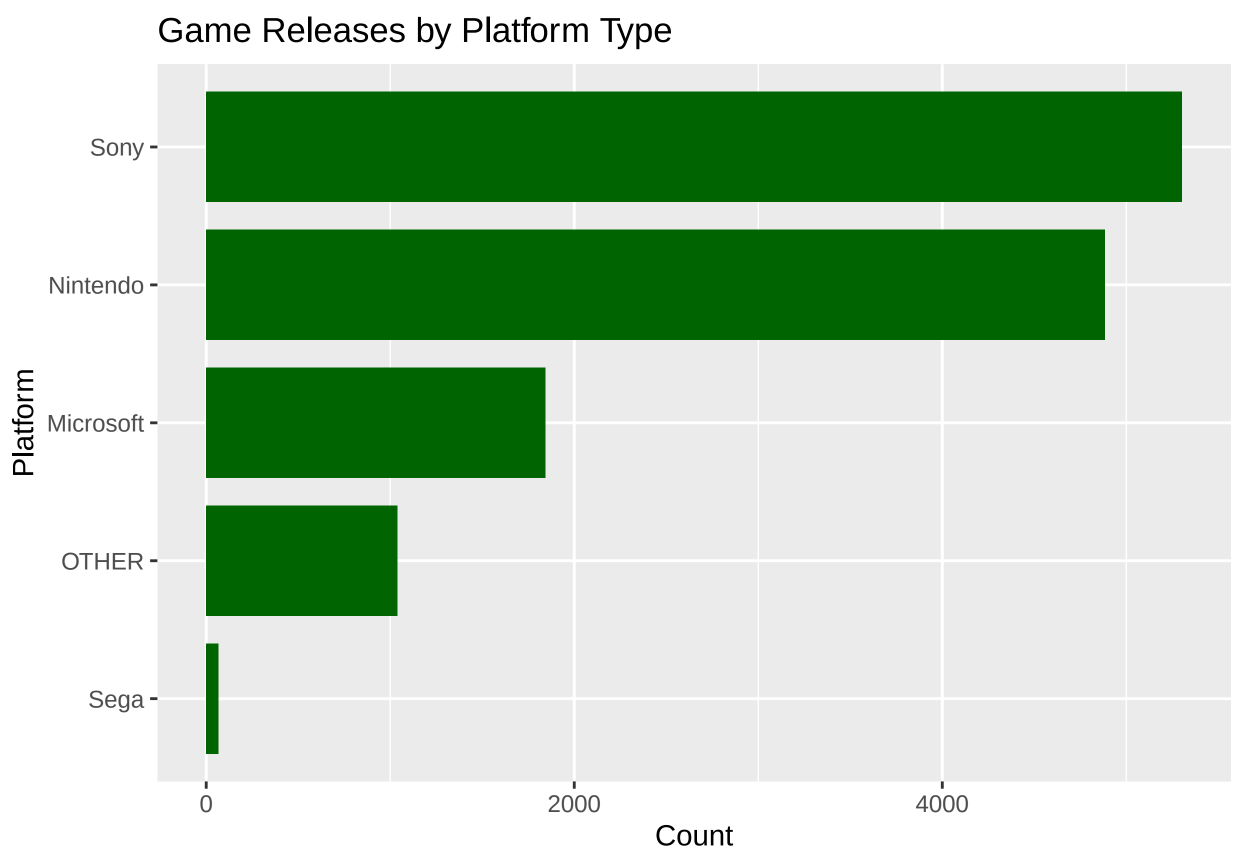
The top 5 platforms by global video game sales and by total number of games released are PS2, X360, PS3, Wii, DS, although in varying order as we can see from the graphs above. PS2 dominates both categories. From this information, we can decipher that these are the most popular platforms/consoles and there seems to be a relationship between the number of games released by platform and the number of units sold.

Platform Type (Manufacturer)

This refers to the companies that produce, manufacture and release the different platforms/consoles.

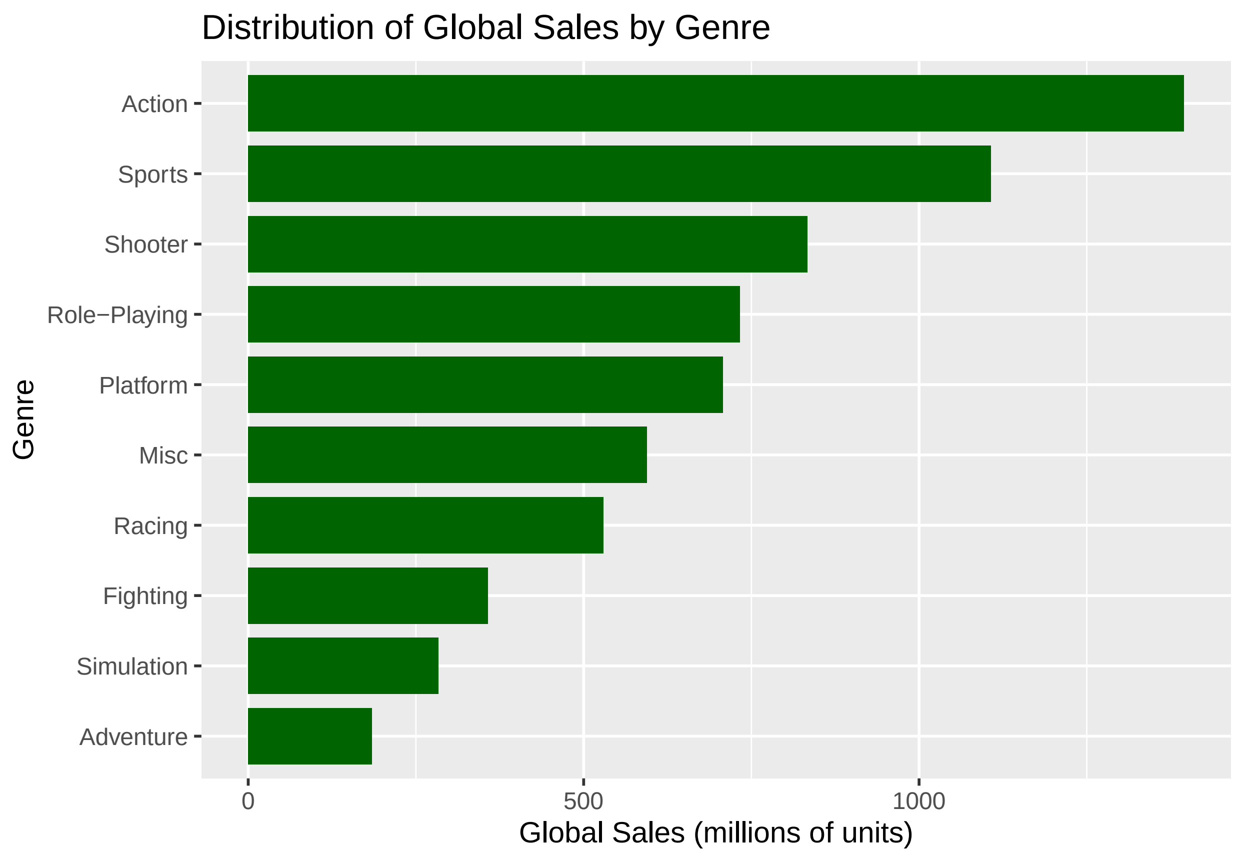


Distribution of Global Sales by Platform Type
Sony
Nintendo
Microsoft
OTHER
Sega
Platform Type
0
1000
2000
Global Sales (millions of units)

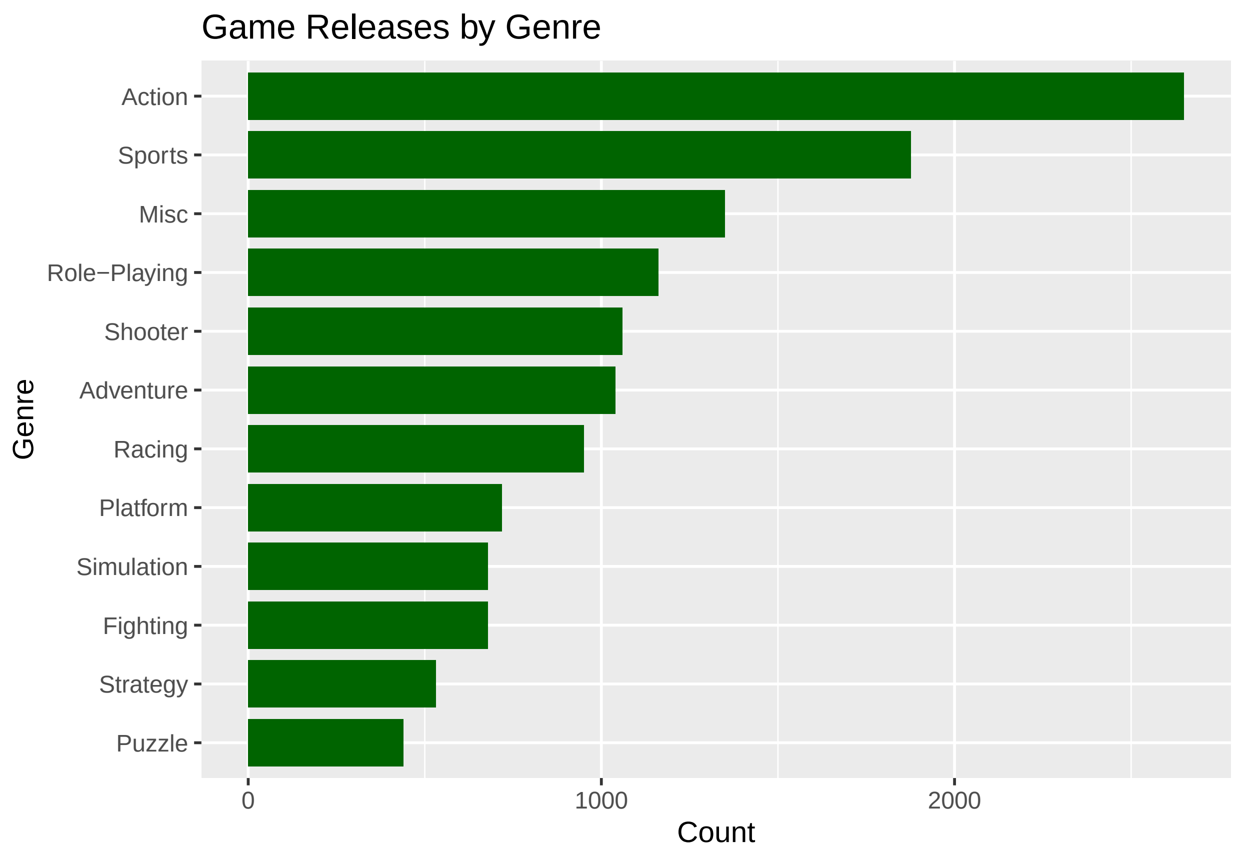


From these graphs we can see that Sony and Nintendo release the most games and also dominate total global video game sales. According to our intuition, these are the most popular platforms for consumers to own and therefore the games released for these platforms perform better in sales.

2.4.3 Genre



Distribution of Global Sales by Genre
Genre
Action
Sports
Shooter
Role−Playing
Platform
Misc
Racing
Fighting
Simulation
Adventure
0
500
1000
Global Sales (millions of units)

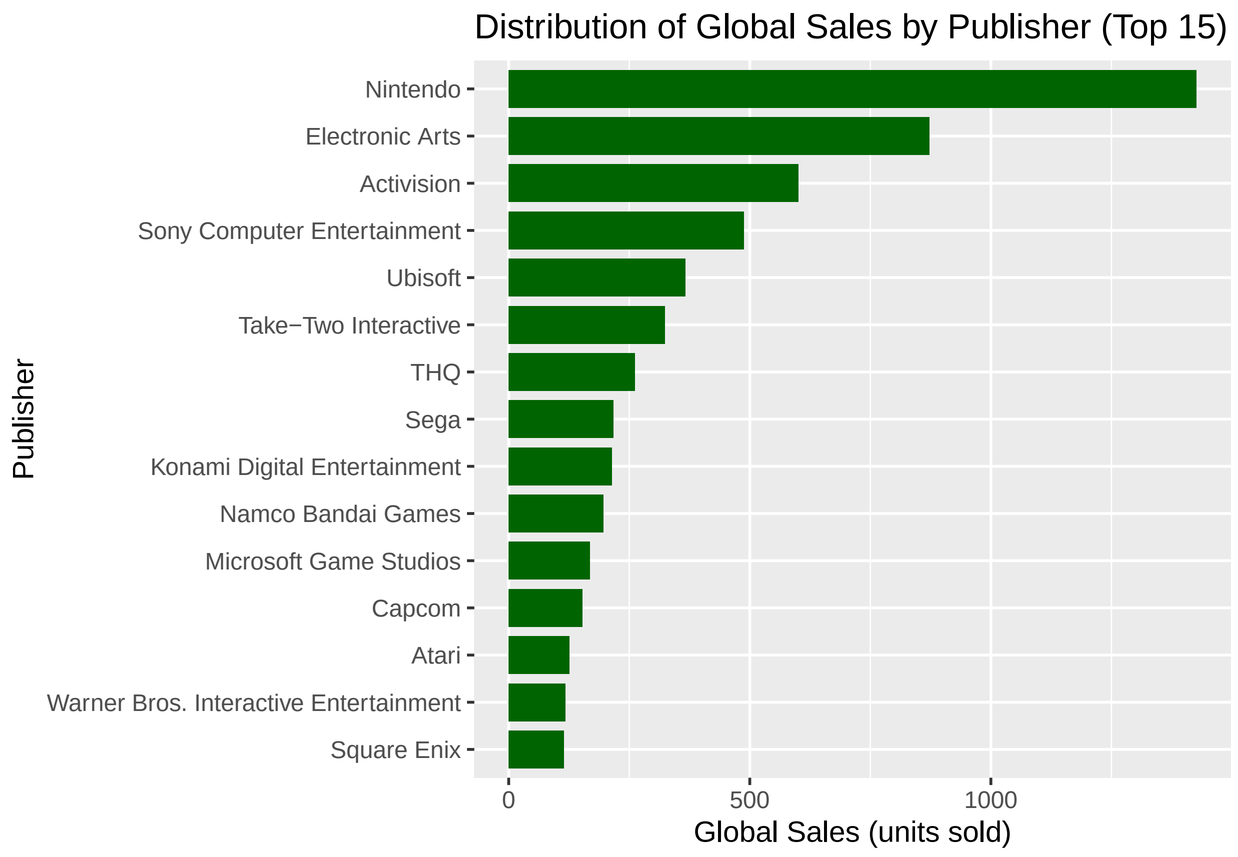


Game Releases by Genre
Genre
Action
Sports
Misc
Role−Playing
Shooter
Adventure
Racing
Platform
Simulation
Fighting
Strategy
Puzzle
0
1000
2000
Count

We can see that the Action, Sports, Shooter, and Role-Playing genres are the most popular. It could be the case that game developers release more games in these categories to match demand and because they're more likely to be successful in terms of sales.

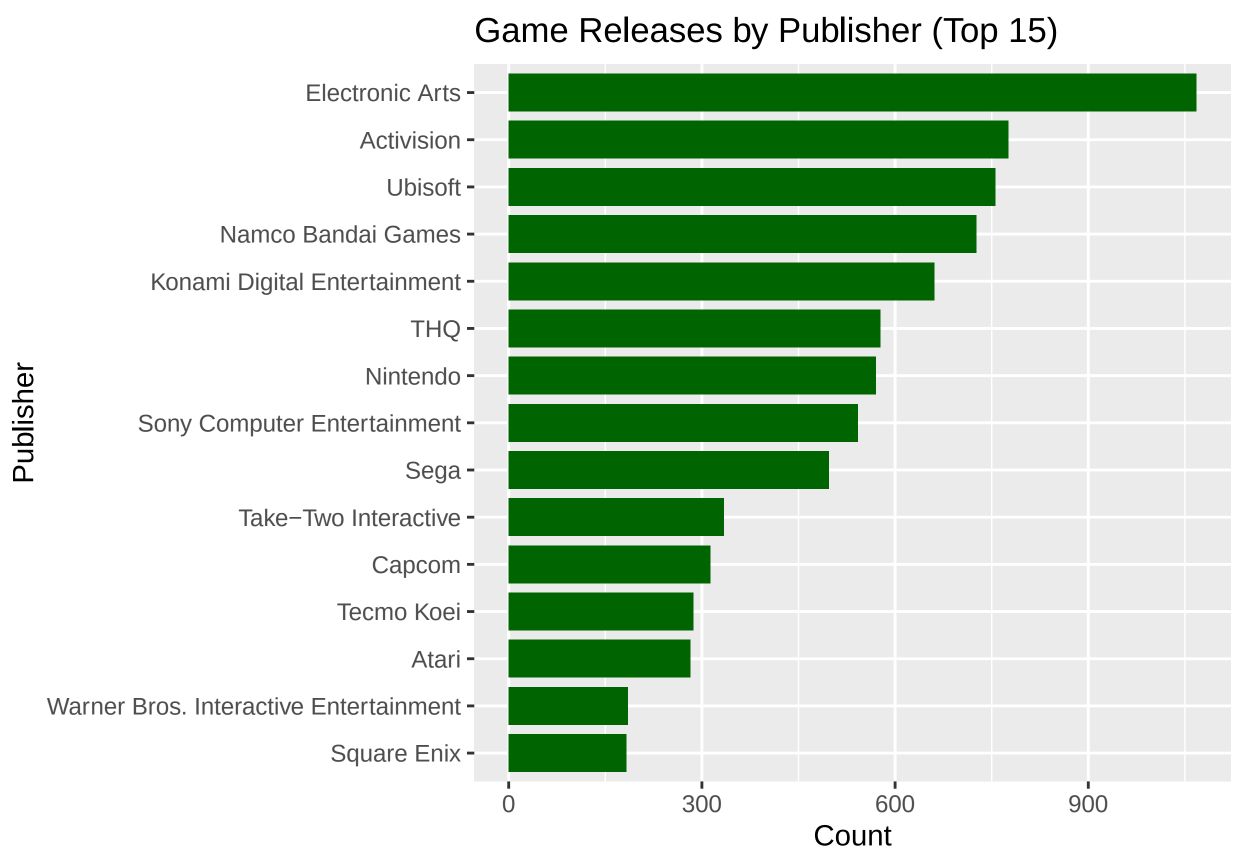
2.4.4 Publisher & Developer

The categories 'Publisher' and 'Developer' seem similar in nature as they refer to the entities involved with the videogames but there are key differences. The main difference is that developers are involved with designing and creating the video game whereas publishers take care of delivering the games to consumers through marketing, sales, and PR[15]. Even though they have separate functions, there is crossover in how they operate. Some developers choose to publish and promote their own video games, while some major publishers have their own developers creating games for them[16].



[15]https://medium.com/@PaulTrowe/the-difference-between-a-video-game-developer-and-publisher-c6038324ee56#:~:text=Usually%2C%20the%20core%20difference,sales%2C%20and%20PR%20of%20it.&text=When%20publishers%20like%20a%2

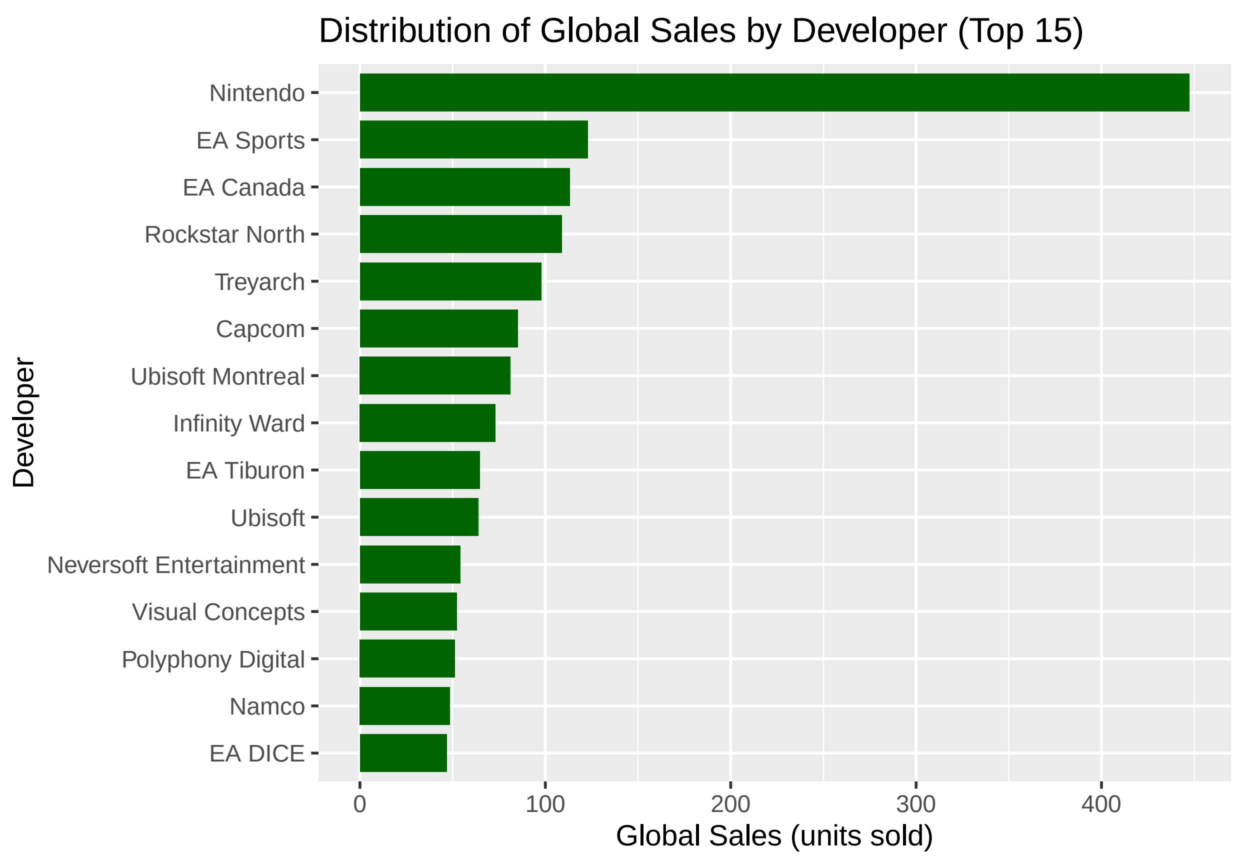
[16]https://medium.com/@PaulTrowe/the-difference-between-a-video-game-developer-and-publisher-c6038324ee56#:~:text=Usually%2C%20the%20core%20difference,sales%2C%20and%20PR%20of%20it.&text=When%20publishers%20like%20a%2

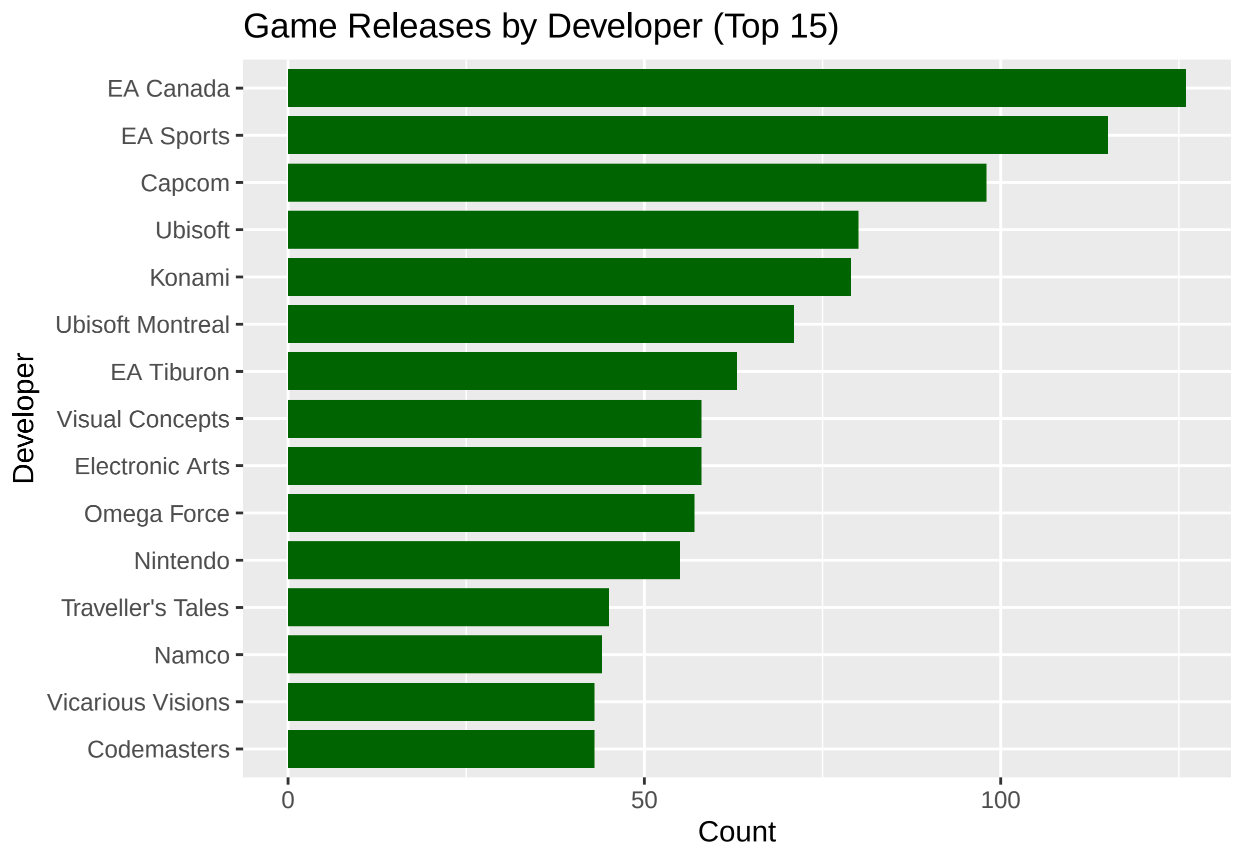


The top 3 publishers by global sales are Nintendo, Electronic Arts, and Activision and the top 3 publishers by number of games released are Electronic Arts, Activision, and Ubisoft. There is a clear overlap here.

Developer	sales
NA	2208.28
Nintendo	448.90
EA Sports	144.10
EA Canada	119.78
Rockstar North	108.93
Treyarch	97.87
Ubisoft	93.21
Capcom	85.72
Ubisoft Montreal	82.22
Infinity Ward	73.16
EA Tiburon	71.75
Visual Concepts	59.40
Konami	57.14
Neversoft Entertainment	54.54
Electronic Arts	52.28

Since the developer column has many NAs, with almost 40% of the data missing, we will use the vgTrain2 dataset (with no NAs) for better visualization of the top developers.


Distribution of Global Sales by Developer (Top 15)
Developer
Nintendo
EA Sports
EA Canada
Rockstar North
Treyarch
Capcom
Ubisoft Montreal
Infinity Ward
EA Tiburon
Ubisoft
Neversoft Entertainment
Visual Concepts
Polyphony Digital
Namco
EA DICE
0
100
200
300
400
Global Sales (units sold)



Developer	percentage_sales
NA	0.3146347892
Nintendo	0.0639590799
EA Sports	0.0205313063
EA Canada	0.0170662031
Rockstar North	0.0155202998
Treyarch	0.0139444757
Ubisoft	0.0132805209
Capcom	0.0122133489
Ubisoft Montreal	0.0117146704
Infinity Ward	0.0104238055

Although Nintendo is not in the top 5 in terms of number of games released, it dominates global sales. In fact, Nintendo accounts for approximately 6% of global video game sales, the largest of any other developer. This number could be even higher since there are many games in the dataset without a developer listed. It is also important to note that we can find Nintendo across features in the dataset, they are platform manufacturers (Game boy, Wii, etc.), publishers, and developers. This makes Nintendo quite unique compared to the other entities in these columns since they design and market games specifically for their own platforms.

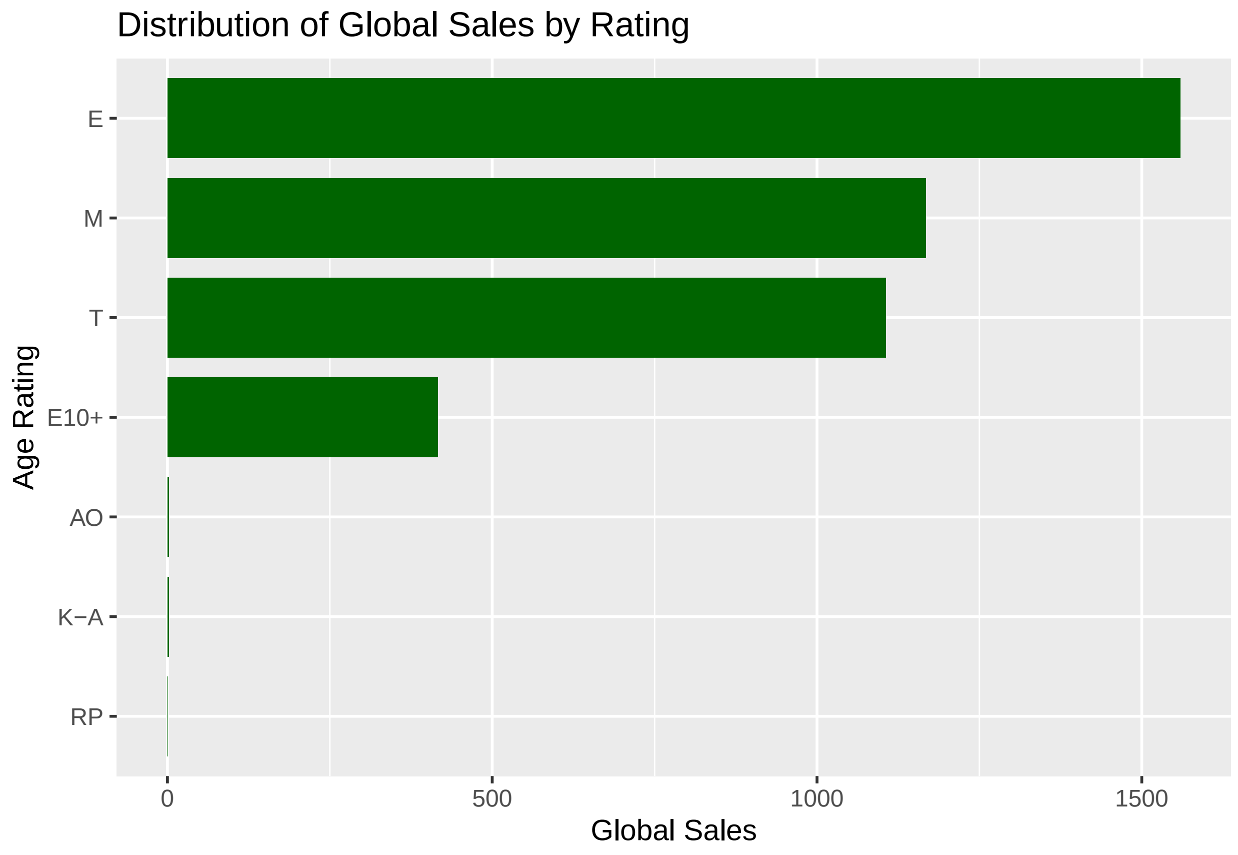
The other top developers are EA Sports (part of Electronic Arts), Ubisoft, Capcom, and Konami, all of whom are also Publishers, which confirms the theory that many companies in the videogame industry do both roles – designing and distributing games.

2.4.5 Rating

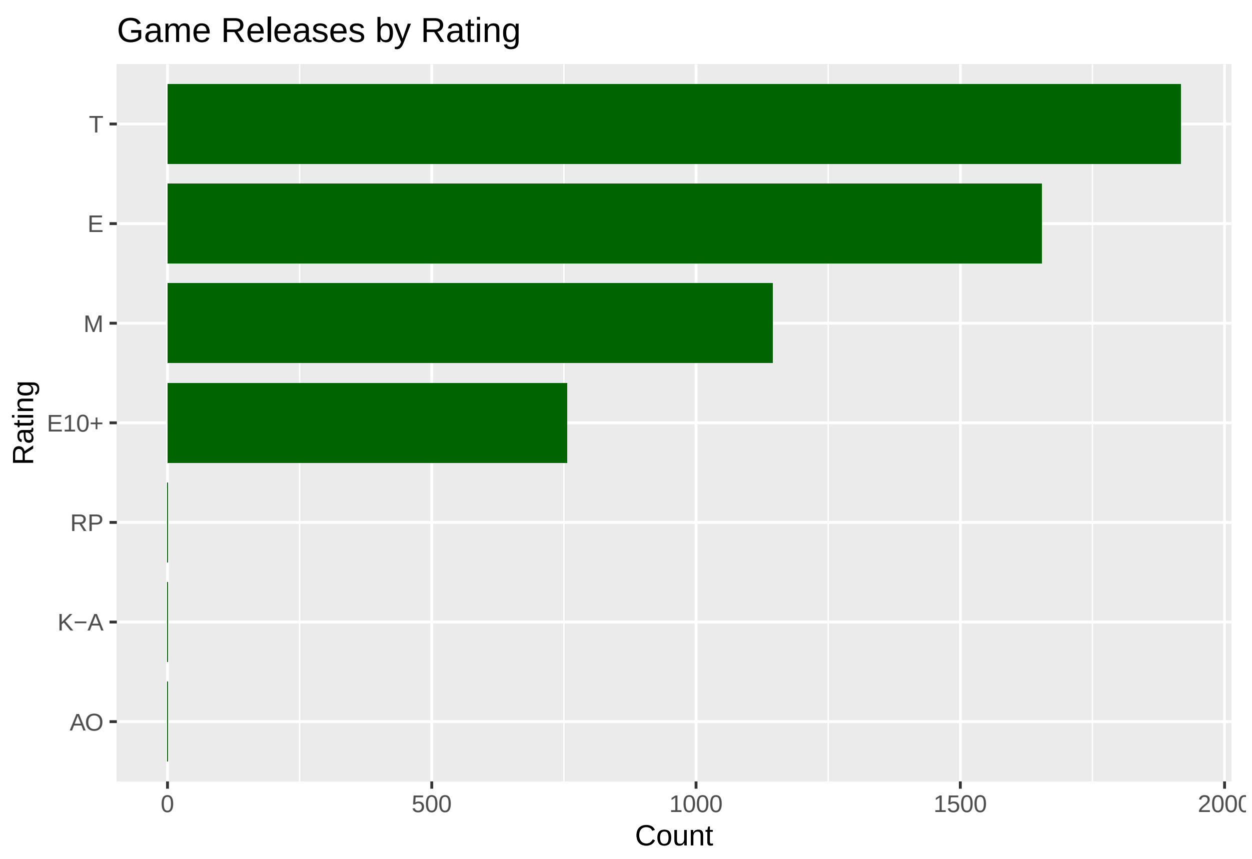
Rating here refers to the ESRB ratings, a regulatory organization that assigns age and content ratings to video games[17]. There are 7 categories of ratings, as follows:

- RP: Rating Pending
- EC: Early Childhood – for a preschool audience. No longer used as of 2018 since all these videogames fall under the E (suitable for everyone) category.
- E: Suitable for everyone. Previously known as Kids to Adults (K-A) until 1998.
- E10+: Suitable for everyone over 10 years of age
- T: Teen – generally suited for those above 13 years of age.
- M: Mature – generally suited for those above 17 years of age.
- AO: Adults Only 18+ - only suitable for those above 18 years of age.

Rating	sales
NA	2224.32
E	1906.63
M	1196.59
T	1183.79
E10+	499.18
K-A	4.33
AO	1.95
EC	1.73
RP	0.03



[17]https://en.wikipedia.org/wiki/Entertainment_Software_Rating_Board

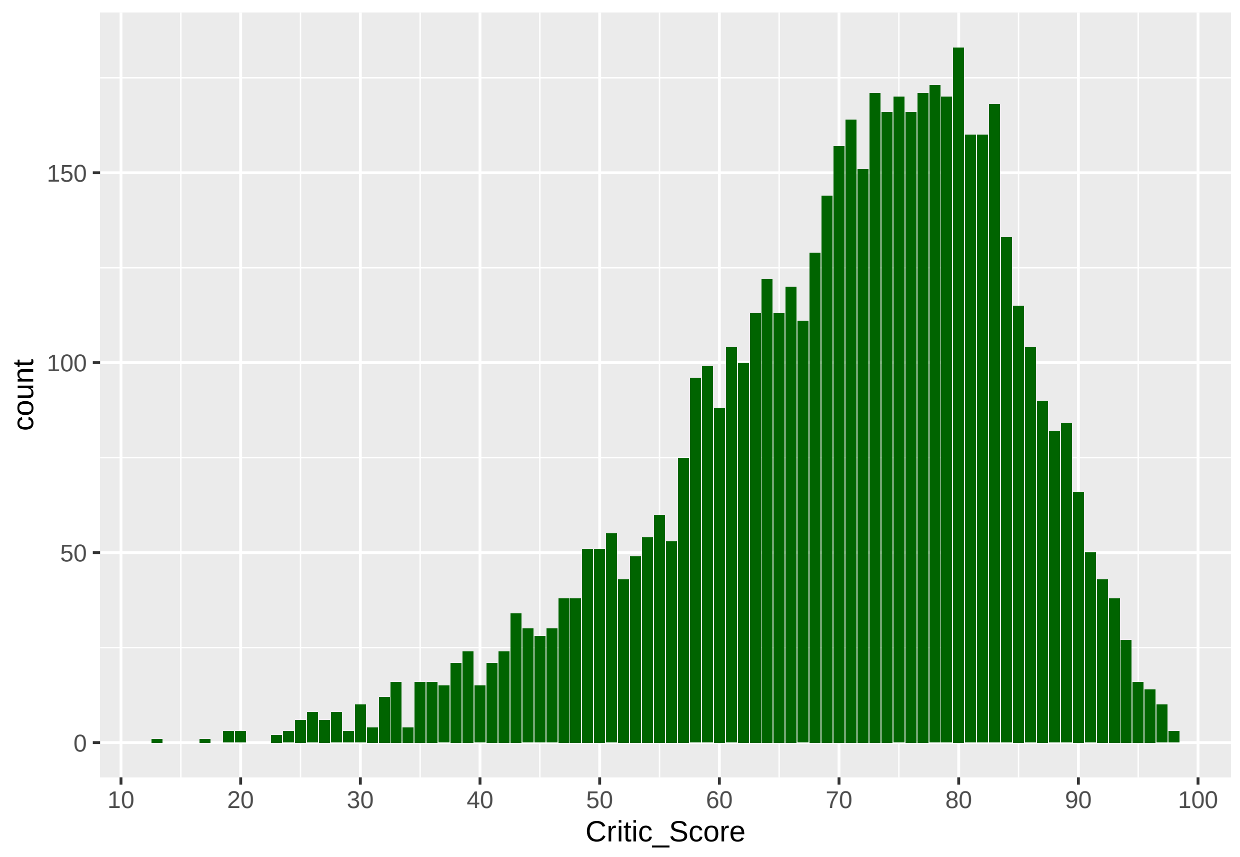


As we can see, there are many video games without a rating. When we work with the dataset with no NAs, we get a better display of the breakdown of the ratings by sales and game releases.

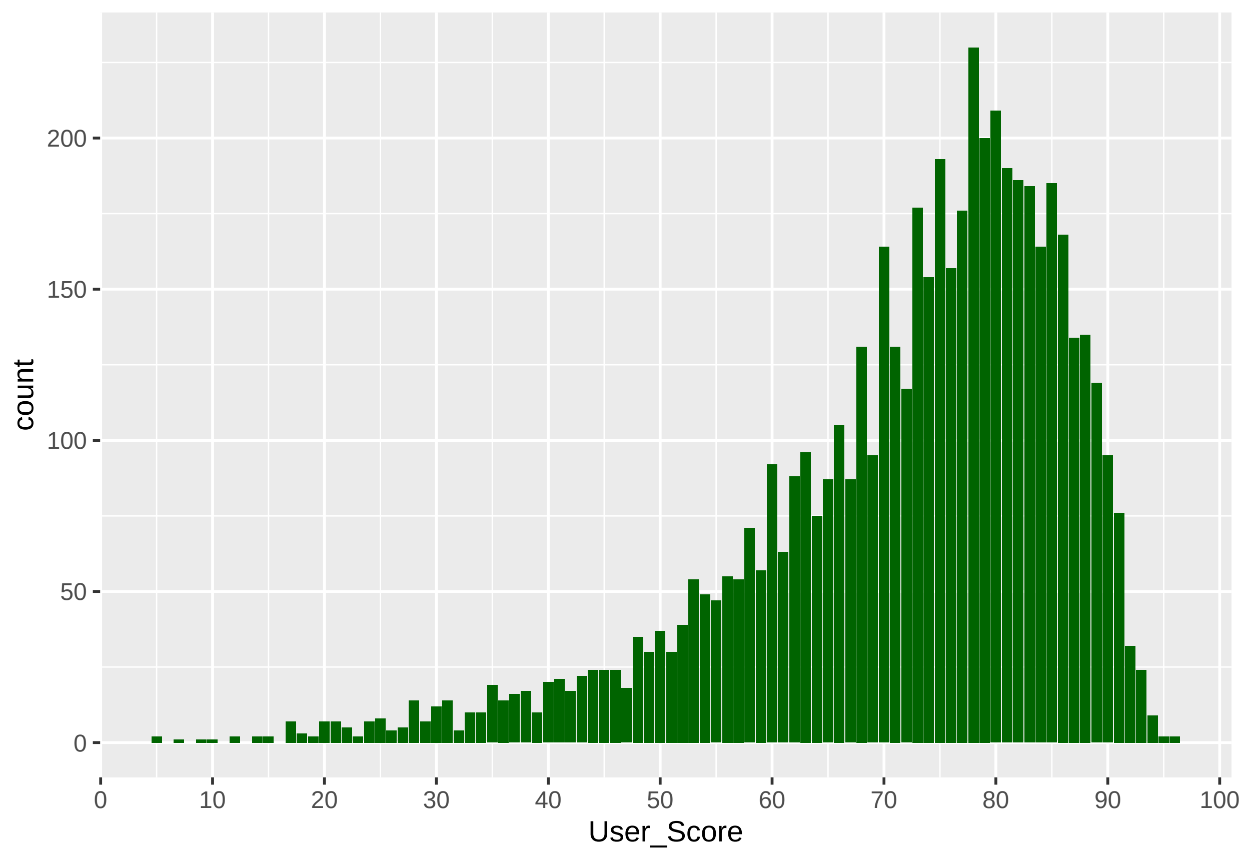
Most games seem to fall into four main categories: E (Everyone), M (Mature), T (Teen), E10+. This is no surprise as adult only video games (AO) are a niche market. Videogames that are appropriate for all age groups seem to fair the best in terms of global sales, which makes sense as they're not constrained by their content. It is interesting to see that the largest number of games are released with a T (Teen) rating, this could be in large part due to the demand for action, shooter, and role-playing games which we discussed earlier in the genre section. These games wouldn't be appropriate for a younger audience.

2.4.6 Critic Score & User Score

Since both features have many NAs, we will refer to the dataset with no NAs.



```
## [1] 70.25808219
```



```
## [1] 71.85515982
```

The distributions of Critic Score and User Score are both left skewed with the average critic score being 70.25 and the average user score being 71.85. This means on average most scores fall between 50-100, with a small portion of games receiving less than 50 points. The distribution of the Critic Score is wider meaning there's more variance across the board.

Next we look at the top 10 games by Critic Score and top 10 games by User Score to check if these are similar. For both lists we are using the vgTrain2 dataset which has no missing values.

Top 10 Games by Critic Score

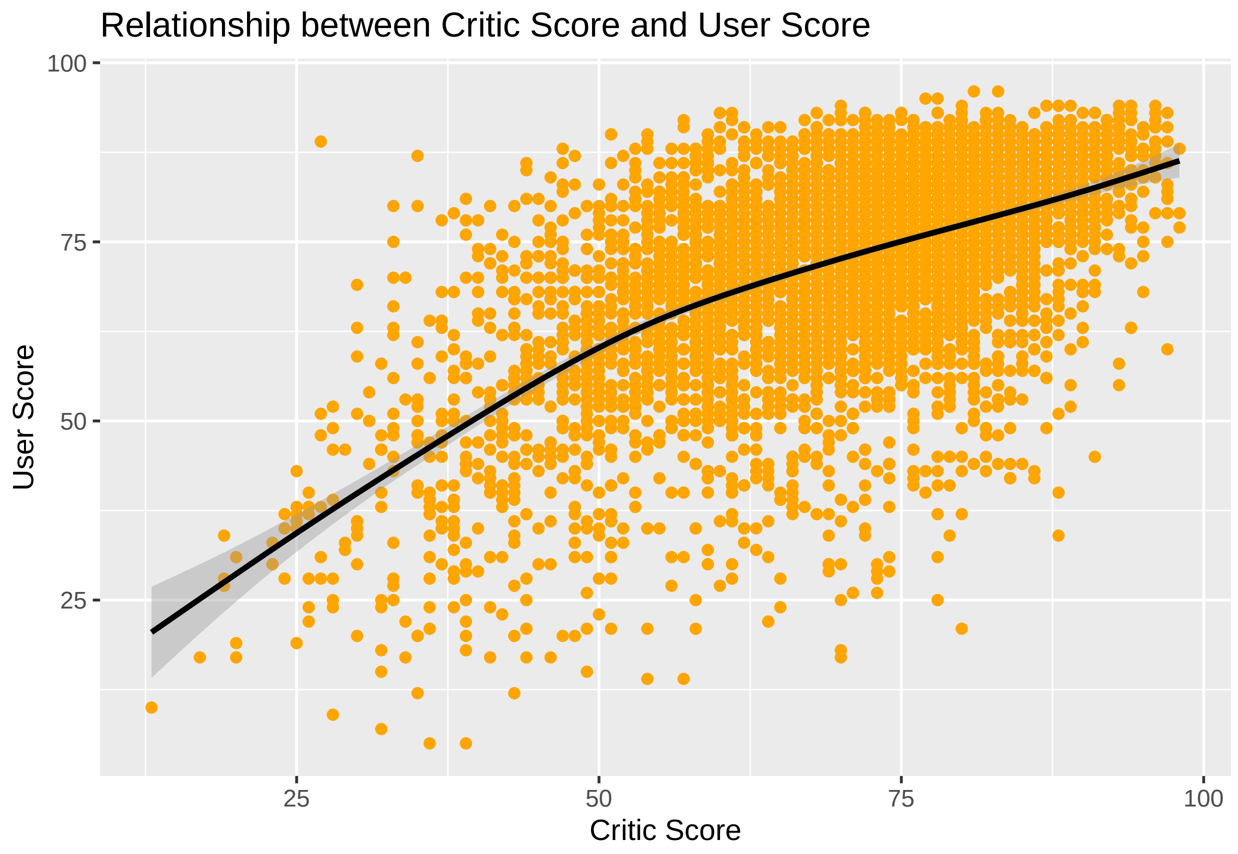
Name	avg
SoulCalibur	98.0
Metroid Prime	97.0
NFL 2K1	97.0
Super Mario Galaxy	97.0
Super Mario Galaxy 2	97.0
Grand Theft Auto V	96.8
Tony Hawk's Pro Skater 2	96.5
Gran Turismo	96.0
Tekken 3	96.0
The Orange Box	96.0
Uncharted 2: Among Thieves	96.0

Top 10 Games by User Score

Name	avg
Boktai: The Sun is in Your Hand	96
Harvest Moon: Friends of Mineral Town	96
MLB SlugFest Loaded	95
Wade Hixton's Counter Punch	95
Advance Wars 2: Black Hole Rising	94
Backyard Baseball	94
Metal Gear Solid	94
Paper Mario: The Thousand-Year Door	94
Rock 'N Roll Racing	94
Shenmue	94
Skies of Arcadia	94

There is no overlap between these top 10s meaning critic hits are not always user favorites.

Next we look at the relationship between critic score and user score to check if there's any correlation between the two variables.



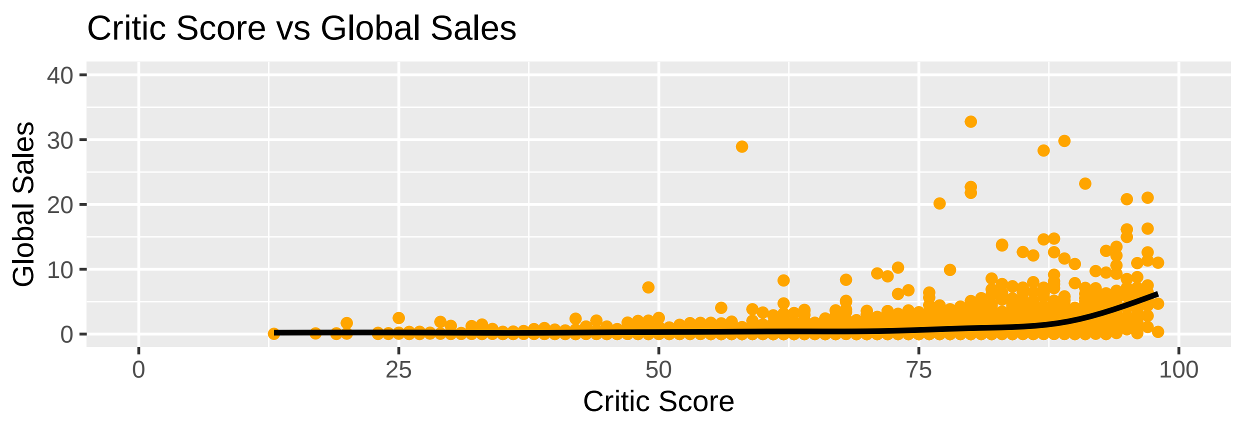
The graph confirms that there is indeed quite a strong positive correlation between the two variables. The correlation function confirms this:

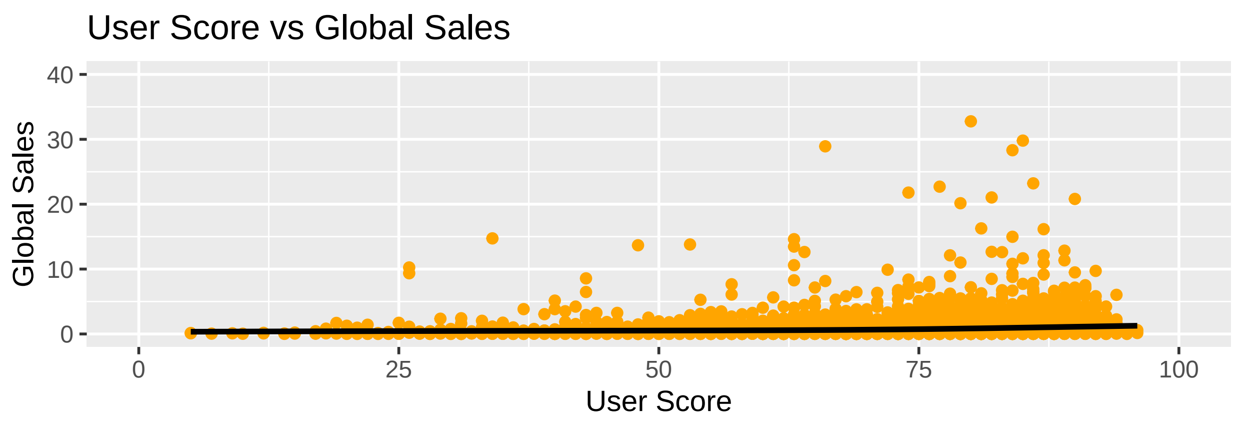
```r
cor(vgTrain2$User_Score,vgTrain2$Critic_Score)
```

```
## [1] 0.5849376946
```

Correlation does not mean causation. However, it does confirm that many games that receive high critic scores also receive high user scores. From the graph we can see that especially for games which received a critic score over 50, the corresponding user score was also higher (above 50).

Next we want to check which variable has a stronger relationship with Global Sales, if any at all.





We can see that although there is a linear relationship between User Score and Global Sales, it is not very strong. Critic Score on the other hand has a stronger effect on Global Sales, especially for very highly scored games. Games that receive a critic score above 85 seem to sell exponentially better. We can confirm that Critic Score has a higher correlation with Global Sales than User Score.

Finally, we look at the average critic scores and user scores by genre.

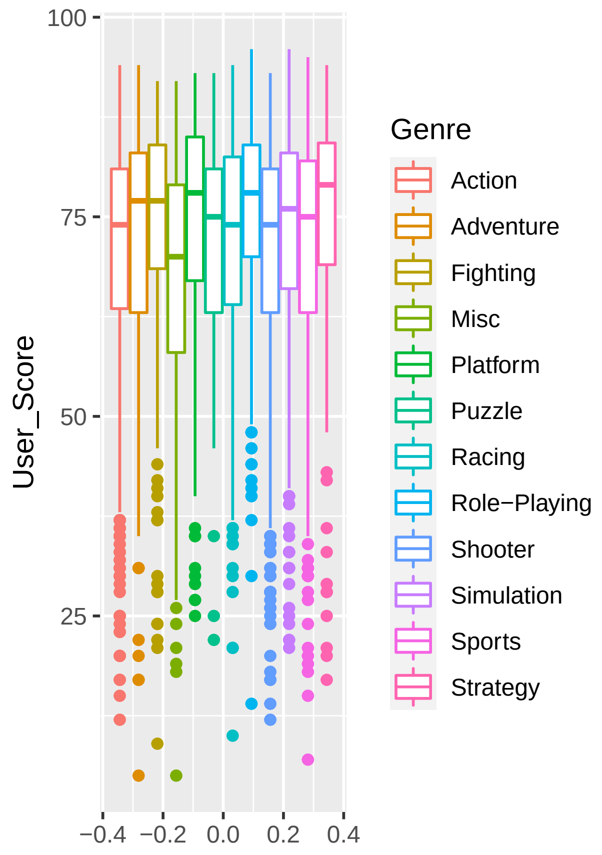
Top Genres by Critic Score

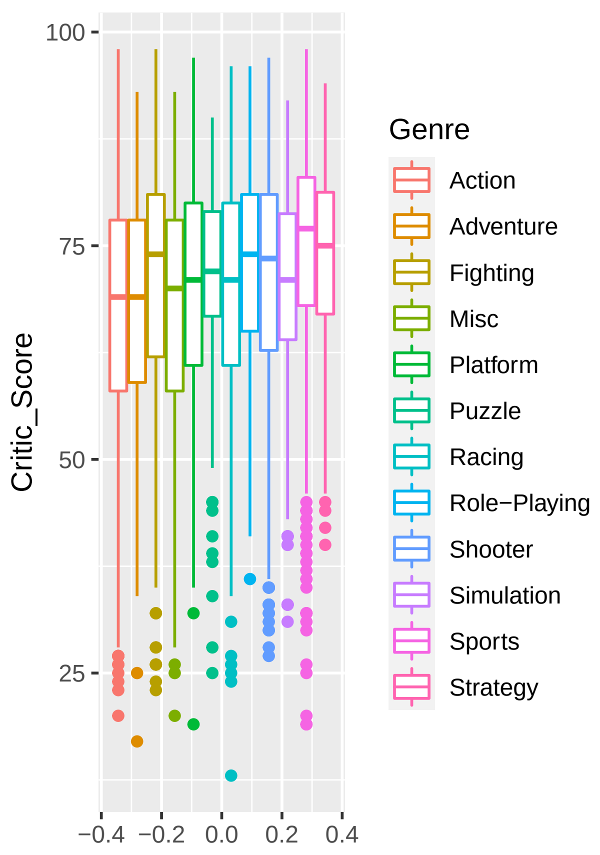
Genre	critic_score
Sports	74.38532110
Strategy	73.67924528
Role-Playing	72.51336898
Shooter	70.80726257
Platform	70.57357357
Fighting	70.36655949
Puzzle	69.75000000
Simulation	69.60869565
Racing	69.39729120
Action	67.75663381
Adventure	66.70707071
Misc	66.62458472

Top Genres by User Score

Genre	user_score
Role-Playing	75.91800357
Platform	74.49249249
Strategy	74.26886792
Fighting	73.53054662
Puzzle	72.07954545
Adventure	71.71717172
Racing	71.61625282
Simulation	71.60869565
Sports	71.25294889
Action	70.83396513
Shooter	70.55167598
Misc	67.60132890

Strategy and Role-Playing appear in the top 3 for both scores.





From the graph we can see that average critic scores are lower than average user scores pretty much across the board. In other words, critics rate movies more harshly than users.

2.4.7 Critic Count & User Count

Critic Count and User Count refer to the number of critics and users that have reviewed each game. We start by looking at which games have been rated the most by critics and users (not to be confused with the highest rated in terms of score). This usually coincides with the popularity of a game.

Top 10 most rated games by Critics

Name	count
Spider-Man 2	252
Grand Theft Auto V	245
Need for Speed: Most Wanted	236
Tomb Raider: Legend	217
Call of Duty: Modern Warfare 2	207
Ghostbusters: The Video Game	195
The Sims 3	194
Mirror's Edge	192
The Elder Scrolls IV: Oblivion	189
X-Men: The Official Game	187

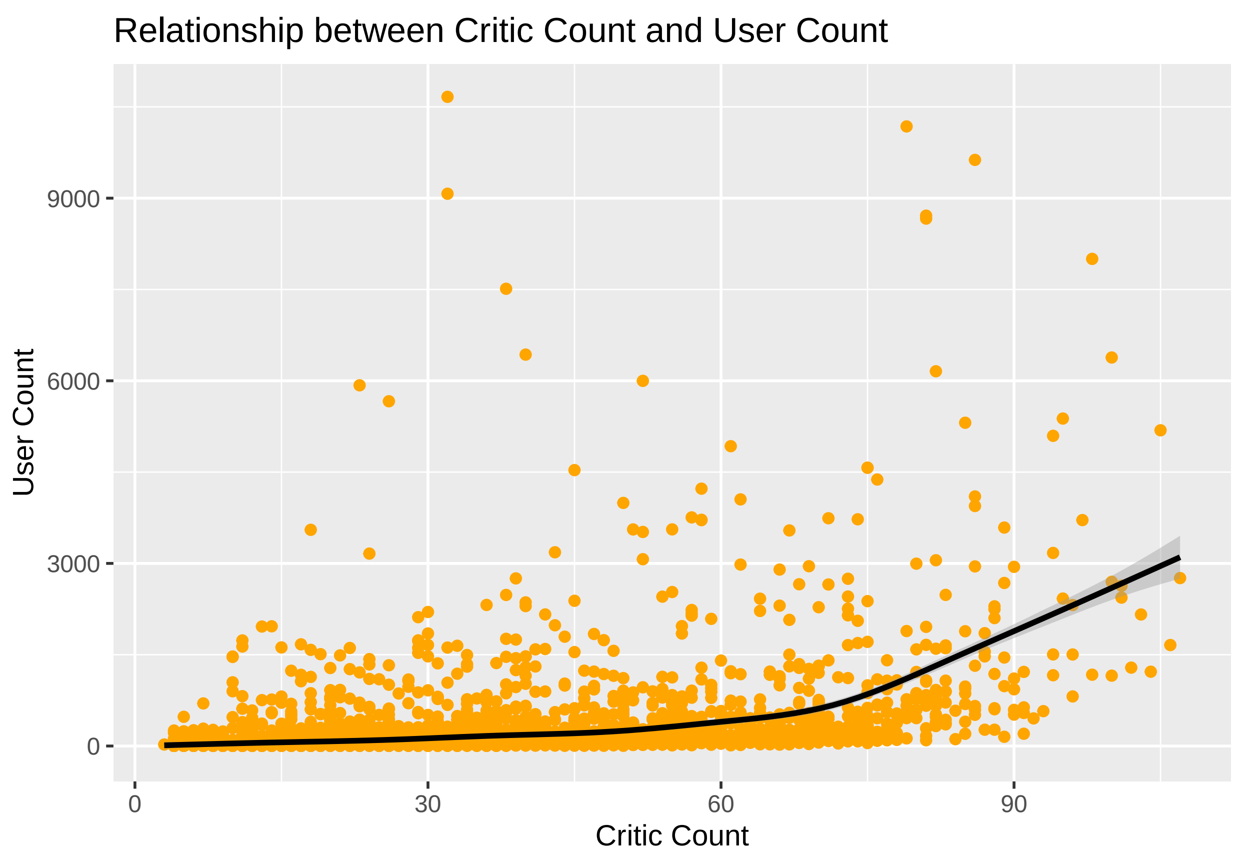
Top 10 most rated games by Users

Name	count
The Witcher 3: Wild Hunt	20844
Grand Theft Auto V	15124
Call of Duty: Modern Warfare 3	14819
Fallout 4	13489
The Elder Scrolls V: Skyrim	12662
Mass Effect 3	11500
Call of Duty: Modern Warfare 2	11201
Diablo III	10186
Battlefield 3	8829
Half-Life 2	8780

As opposed to the top games by critic and user scores which saw no overlap, there is definitely some overlap between these two lists. This could be in part due to the fact that games that receive a lot of hype tend to be rated more often as there is a larger customer base playing these games. Furthermore, games that are rated by many critics may have an effect on the number of consumers buying the game and therefore rating them. We can also see that some of these top 10 games are franchises with multiple sequels meaning there is already a loyal customer fan base who are more likely to buy and review newer editions.

We will check some of our assumptions below.

Is there a relationship between Critic Count and User Count?



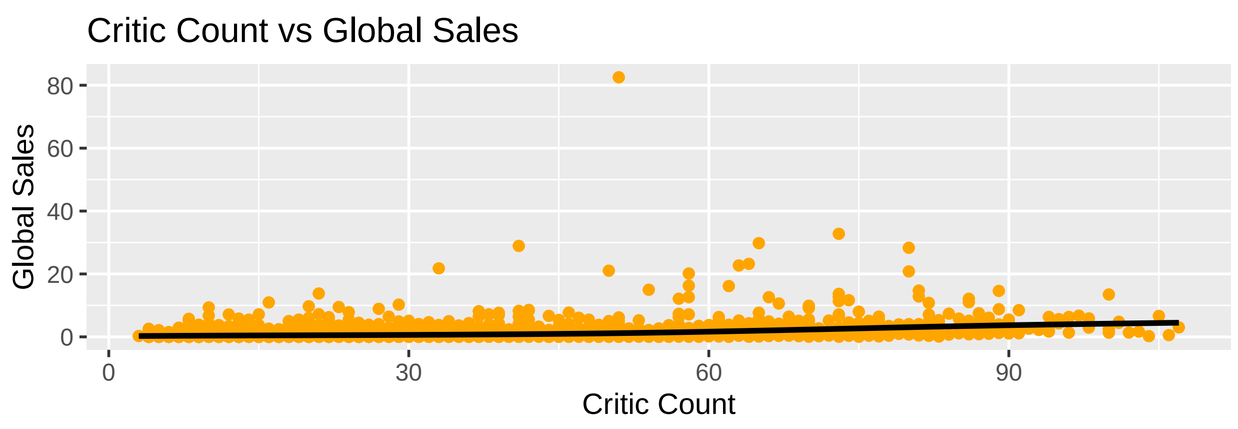
```
#correlation between user count and critic count
cor(vgTrain2$Critic_Count,vgTrain2$User_Count)
```

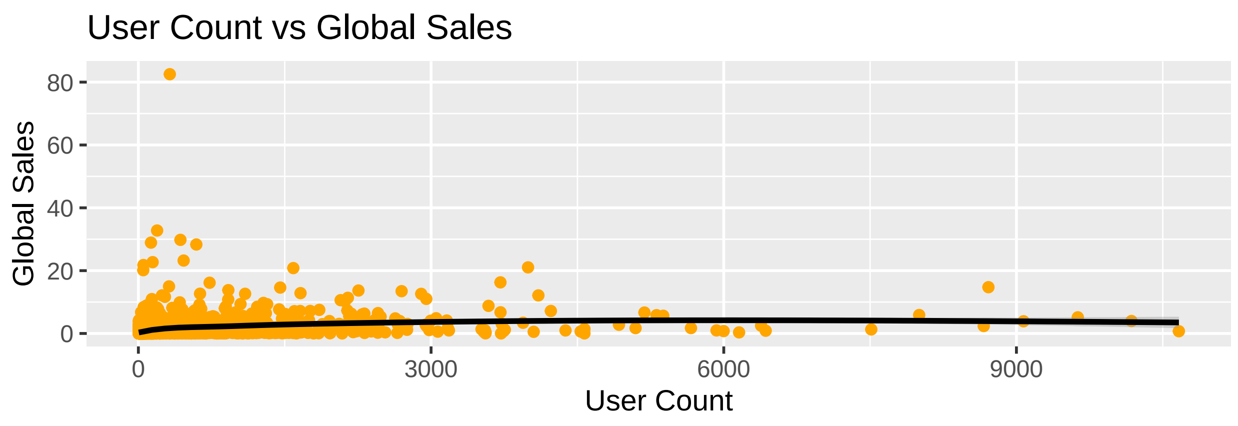
```
## [1] 0.3753491786
```

There is indeed a positive linear relationship between the two variables, and beyond a certain point the relationship only gets stronger. In other words, games that have been rated by a large number of critics (80+) are also likely to have also been rated by more users. Because correlation does not imply causation we do not know which variable has the greater effect on the other, or if there is any effect at all. Are critics more likely to review games that are hyped and in demand by customers? Or are customers more likely to buy and therefore review games that have been rated by many critics?

Are games that are rated more often likely to have higher sales or vice versa?

We will look at the relationship between these variables and Global Sales to see if we can find more answers.





We can see that both variables (Critic Count and User Count) do have a positive linear relationship with Global Sales, however neither relationship seems to be strong. Furthermore, because these variables are on different scales, it is difficult to compare them visually. Instead we will look at their correlations with Global Sales.

```
# Correlations between the count variables and global sales
cor(vgTrain2$Critic_Count,vgTrain2$Global_Sales)
```

```
## [1] 0.2867149652
```

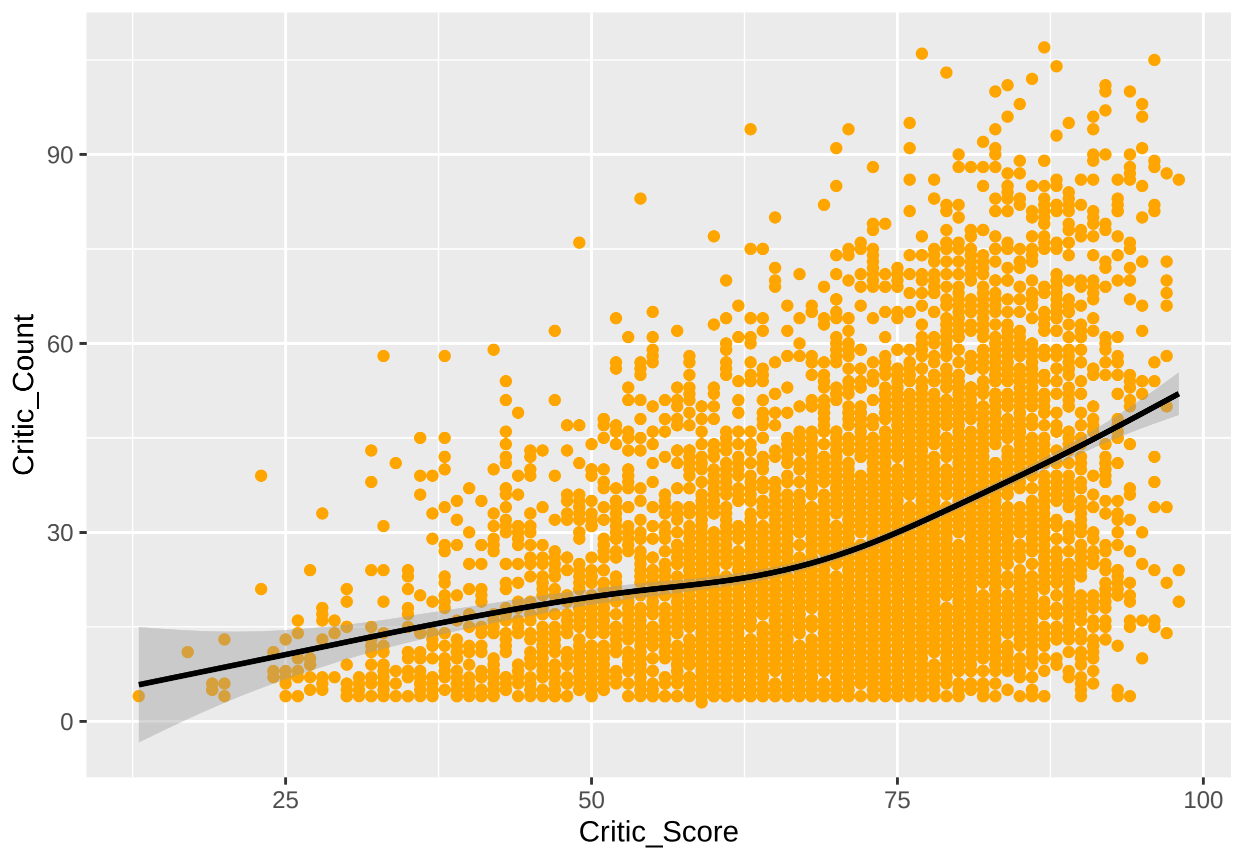
```
cor(vgTrain2$User_Count,vgTrain2$Global_Sales)
```

```
## [1] 0.2638869759
```

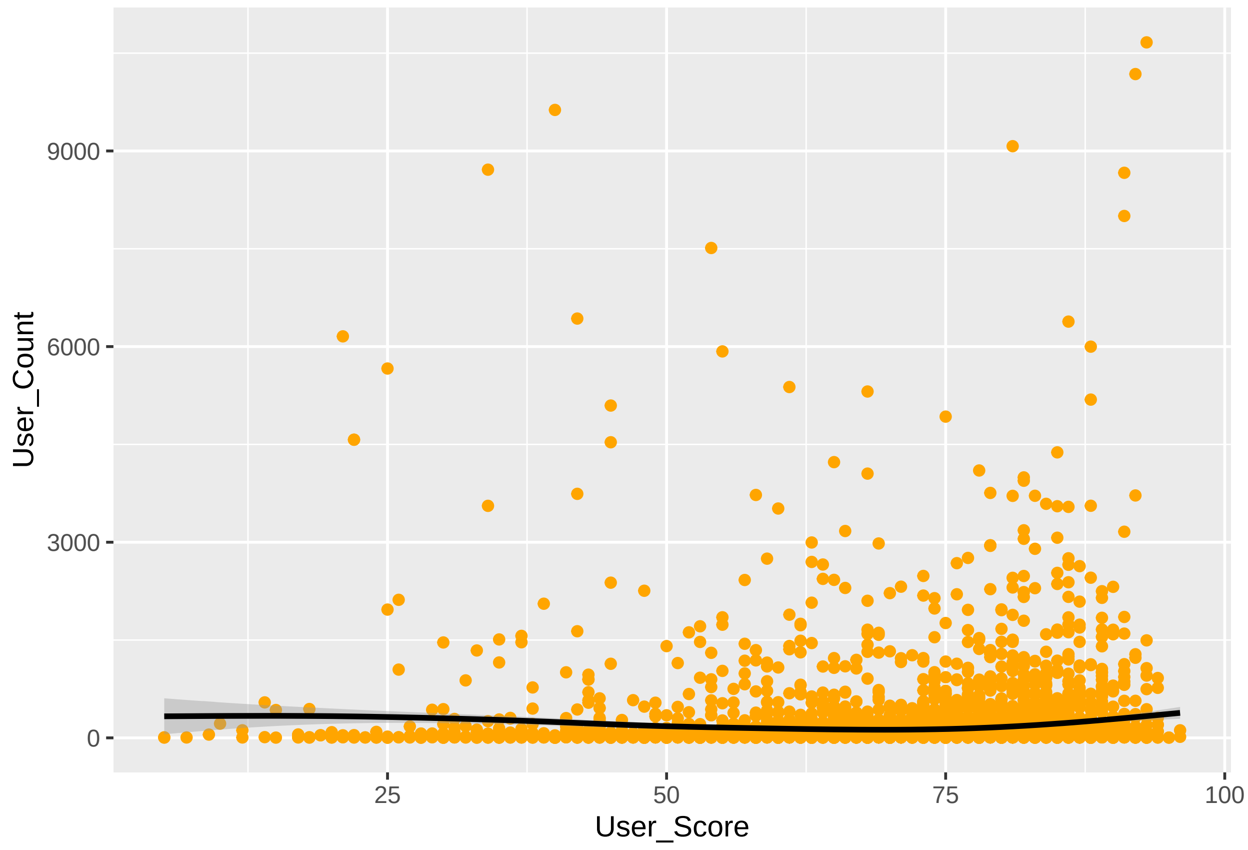
These numbers confirm that there is a relationship between both these variables and Global Sales, however, not a very strong one. Critic Count has a slightly higher correlation with Global Sales than User Score. Again, it is not clear at this stage whether a game that is rated by more critics and users tends to sell more units or the other way around, i.e. are games that are selling a large number of units likely to be rated by more critics and users?

Are highly rated games likely to be reviewed by more critics and users?

We will look at whether Critic Count and User Count have any relationship with the scores given to video games, i.e. Critic Score and User Score.



```
## [1] 0.3949259241
```



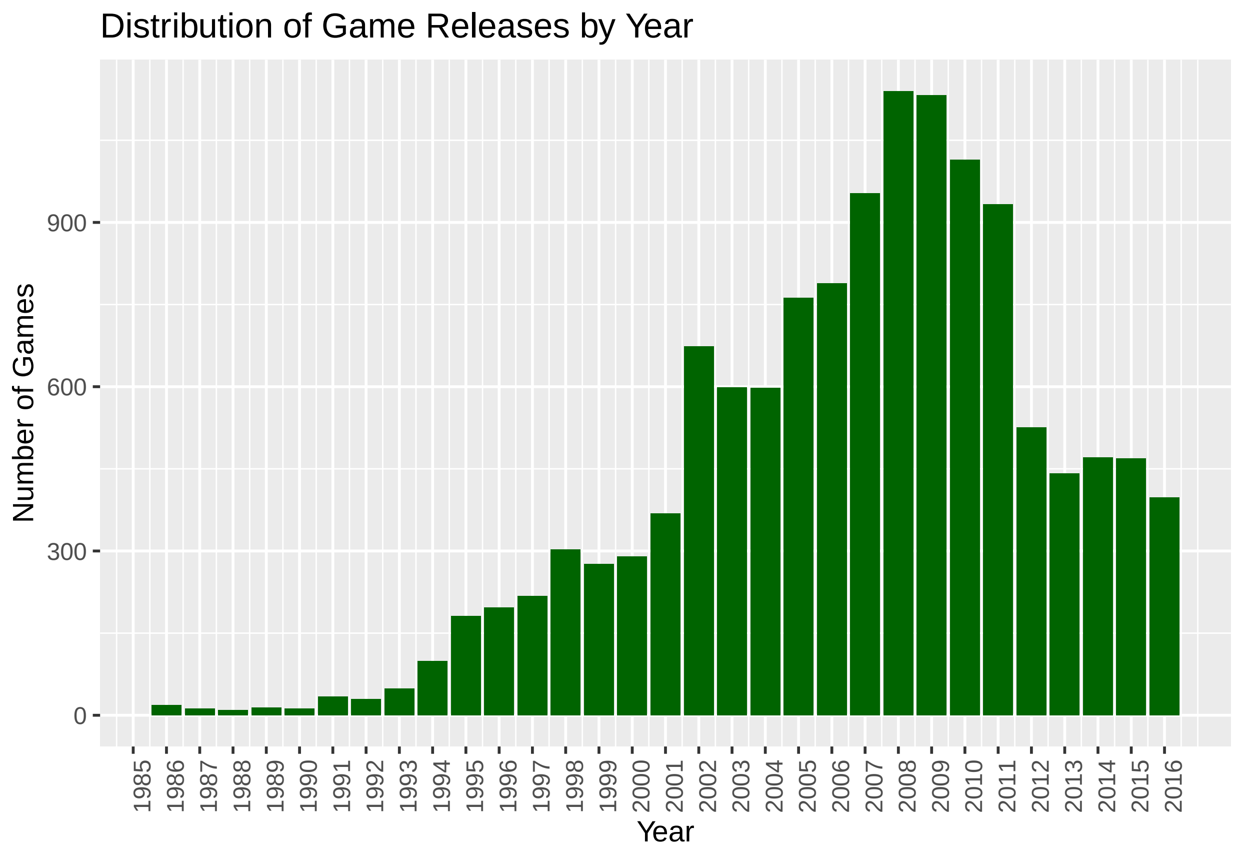
```
## [1] 0.007855419303
```

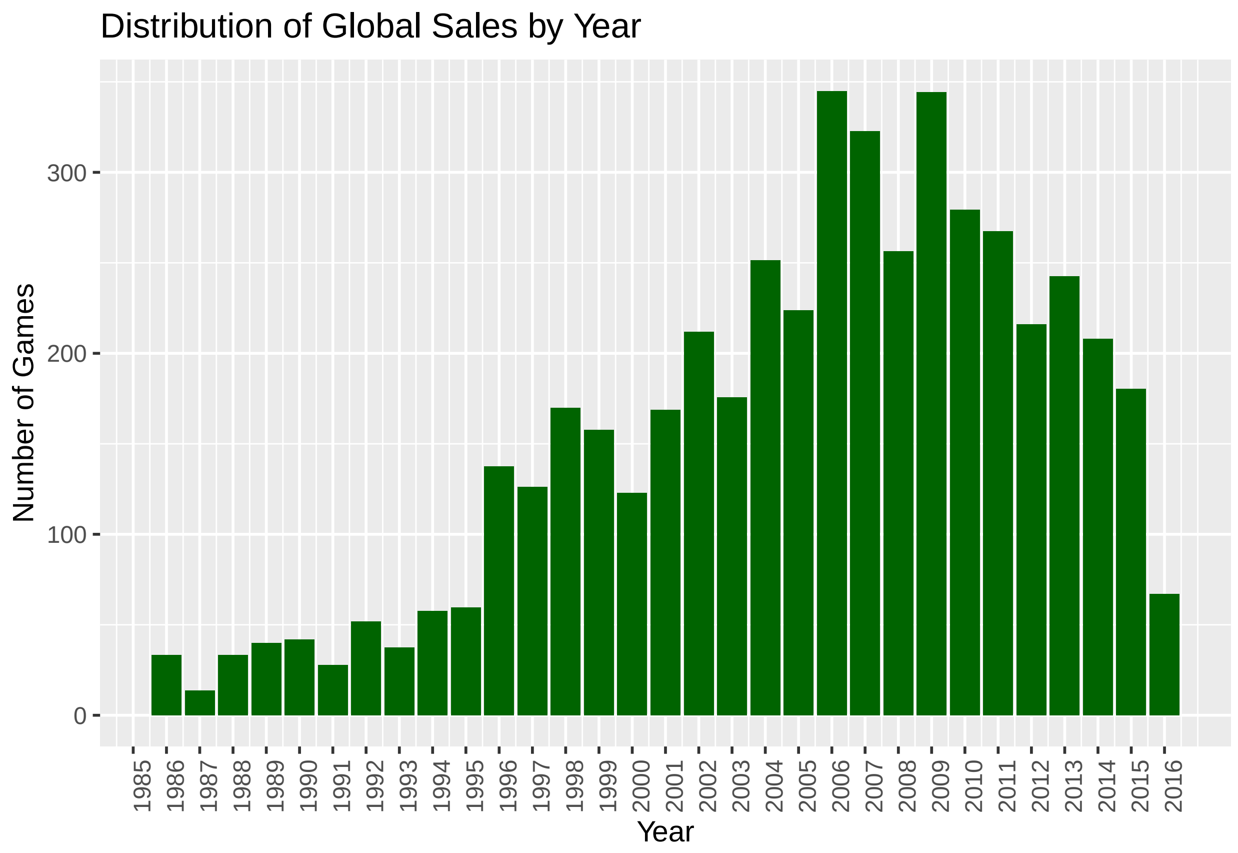
Critic Score and Critic Count have a significant positive relationship which means that critically acclaimed games are likely to be reviewed by more critics. We can't confirm that one causes the other but our intuition says that this could be evidence of a domino effect whereby videogames that start receiving high critic scores gain more acclaim and popularity, prompting even more critics to review them.

There is no significant relationship between User Score and User Count. This seems slightly odd but perhaps it is because highly rated games (high scores) are not always the most popular or said in a different way, popular games are not always well rated. A game could be very popular, thus reviewed more often, but not necessarily receive high scores and vice versa. This is similar to the movie industry in that some movies receive high ratings but are not watched by a large audience, like independent films, and some movies that are watched by millions (blockbusters) are not highly rated.

2.4.8 Year & Age

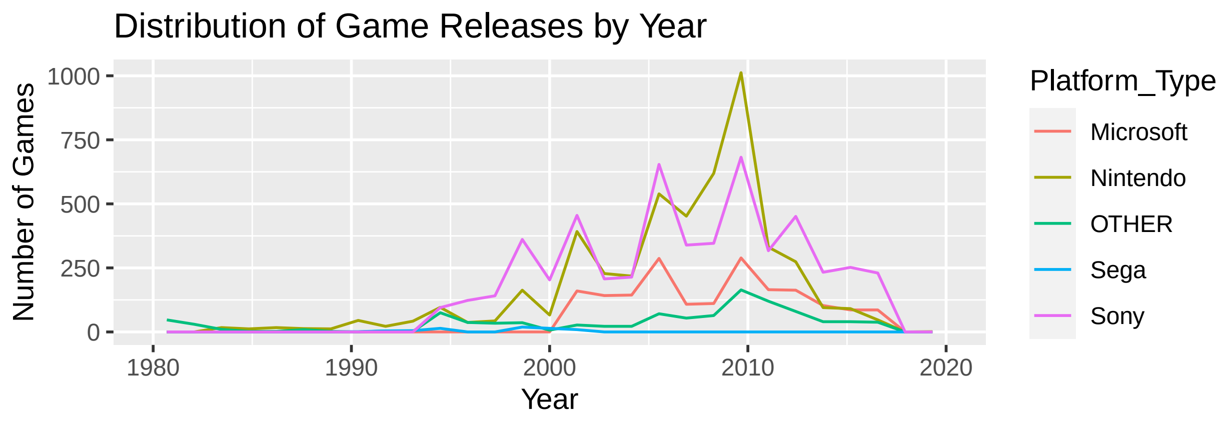
Year refers to the year the game was released. Age is a new feature we created that tells us how old each game is.

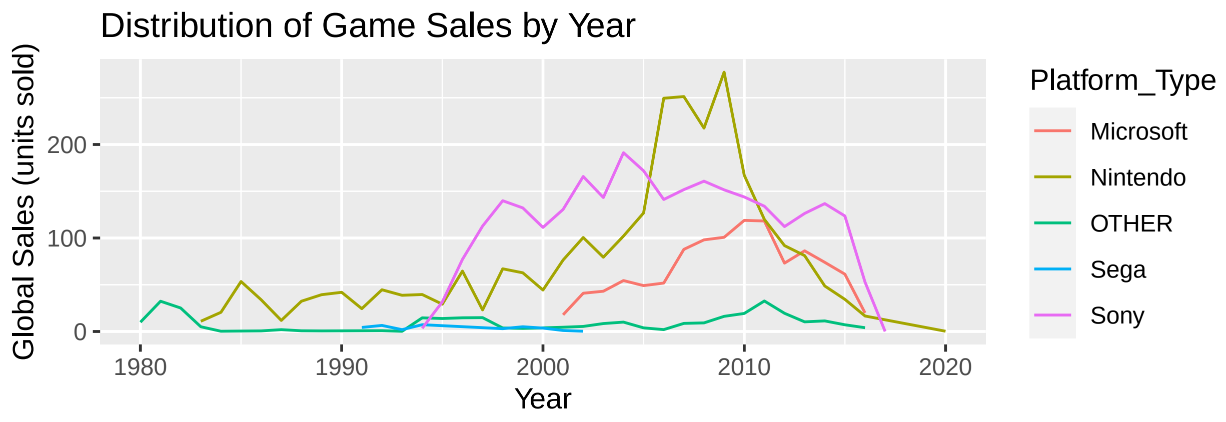

Distribution of Game Releases by Year
Number of Games
0
300
600
900
Year
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016



Since the year 2000, video game releases have steadily increased year on year until 2009 when it started decreasing. 2008 saw the highest number of games being released. This decrease over the past decade could be attributed to the popularity of video games on other platforms such as PCs and mobile phones, which are not covered extensively in this dataset.

When looking at global sales throughout the years, again we see a similar steady increase from 2000 until 2009. Both these trends coincide with the release of the Playstation 2 and Xbox consoles in the early 2000s which had a major impact on the video game industry.

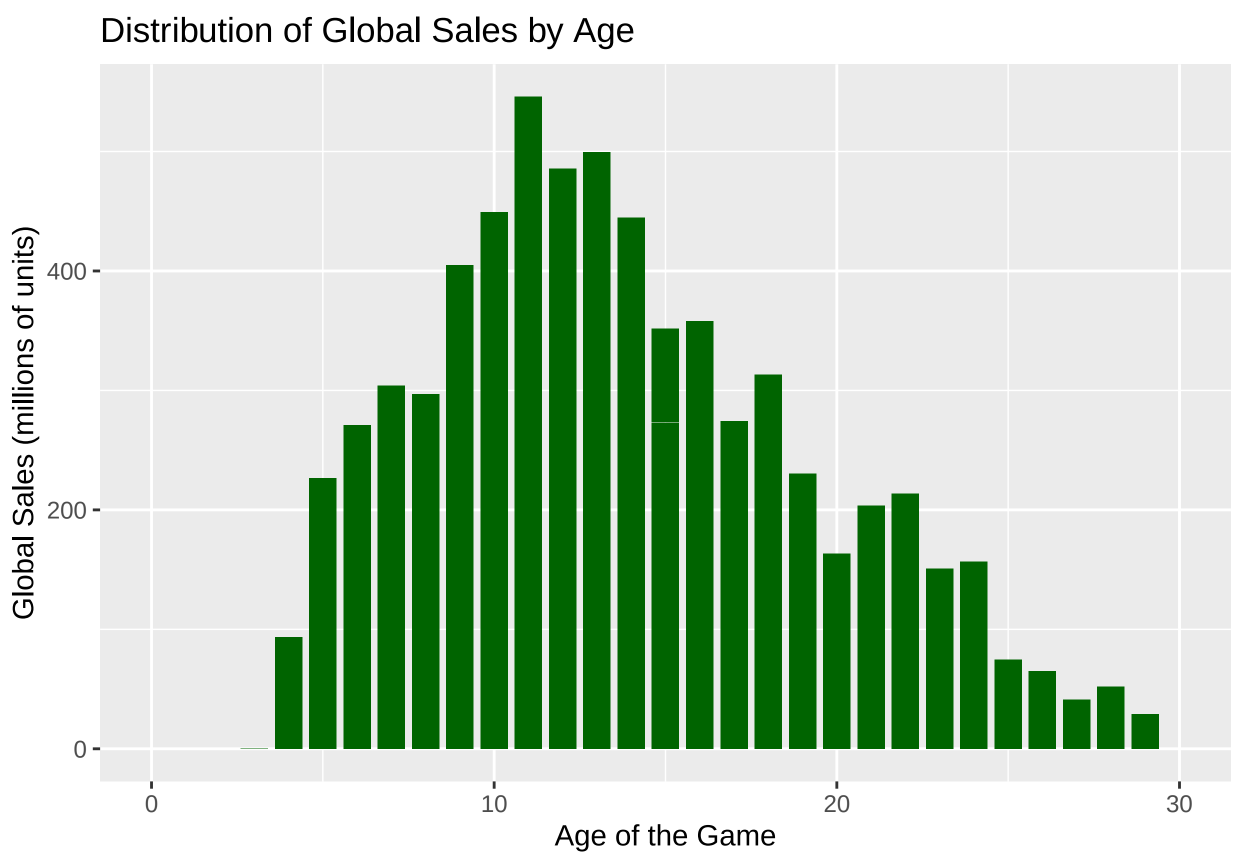




When looking at both these variables (game releases and global sales) by platform type, it is interesting to see that Nintendo had exponential growth between 2000-2010, followed by a steady decline afterwards. For the other platforms, their game releases and global sales seem to follow a similar pattern (i.e. they are likely to sell more games in years when they release more games).

The average age of games in the dataset is:

```
## [1] 13.50312119
```



Most games in this dataset have been around for over a decade. This confirms our intuition that some games are played even years after their release because they are nostalgic or timeless in nature (eg. Tetris).

When looking at the correlation between the age of the game and global sales, we find that there isn't much of a relationship between the two variables.

```
cor(vgTrain$Global_Sales,vgTrain$Age)    #correlation between sales and age of the game
```

```
## [1] 0.07168095669
```

2.5 Feature Selection

In this section, we will aim to select the most relevant features to feed into our machine learning algorithms. Having too many features increases model complexity, making them run too slowly, and may lead to over-fitting, thus producing suboptimal results. There are many different techniques one can adopt to do feature selection. For the purposes of this project, we will be using three methods: correlation, Boruta, and RFE (Recursive Feature Elimination).

Before we go on to feature selection, we will first remove a couple of columns we will no longer be using. We are removing the Name column because it won't be necessary for our model and has too many categories. We are removing the Year column because it is somewhat redundant with the Age column. For feature selection to work effectively, we will be using the dataset with no missing values (vgTrain2).

```
##  [1] "Platform"     "Genre"        "Publisher"     "Global_Sales"
##  [5] "Critic_Score" "Critic_Count" "User_Score"    "User_Count"
##  [9] "Developer"    "Rating"       "Age"           "Platform_Type"
```

2.5.1 Correlation

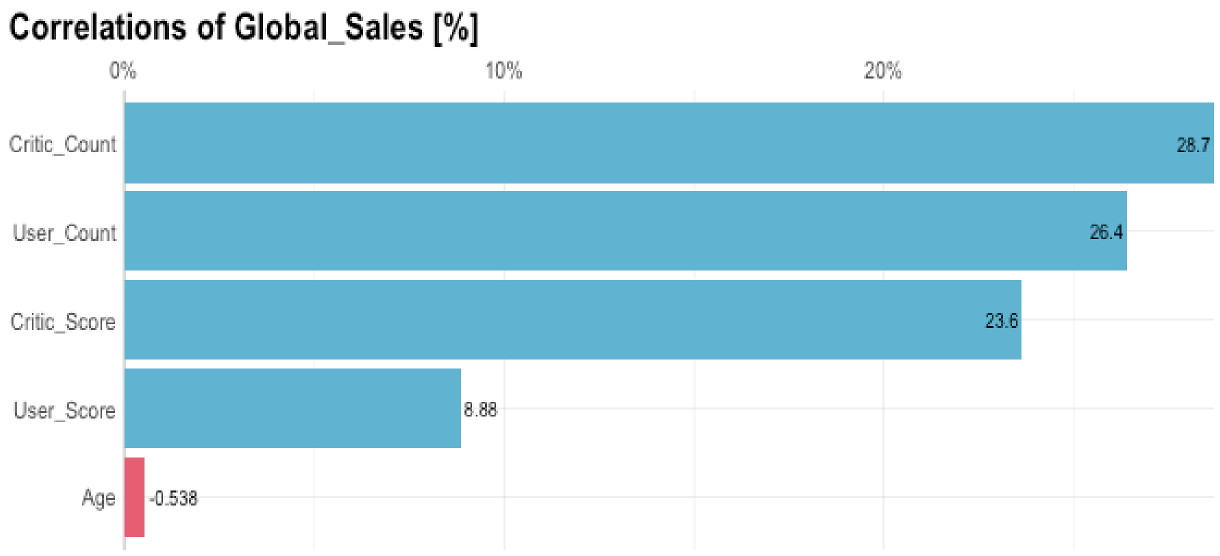
We will first start with one of the simplest feature selection techniques which is to look at the correlations between features and with the target variable (Global Sales). If any two independent variables have a very high correlation (>0.75) with each other, one of them will have to be removed as their effect would be redundant. We will also look at which features have the highest correlation with the target variable as these could help us make better predictions during the modeling stage.

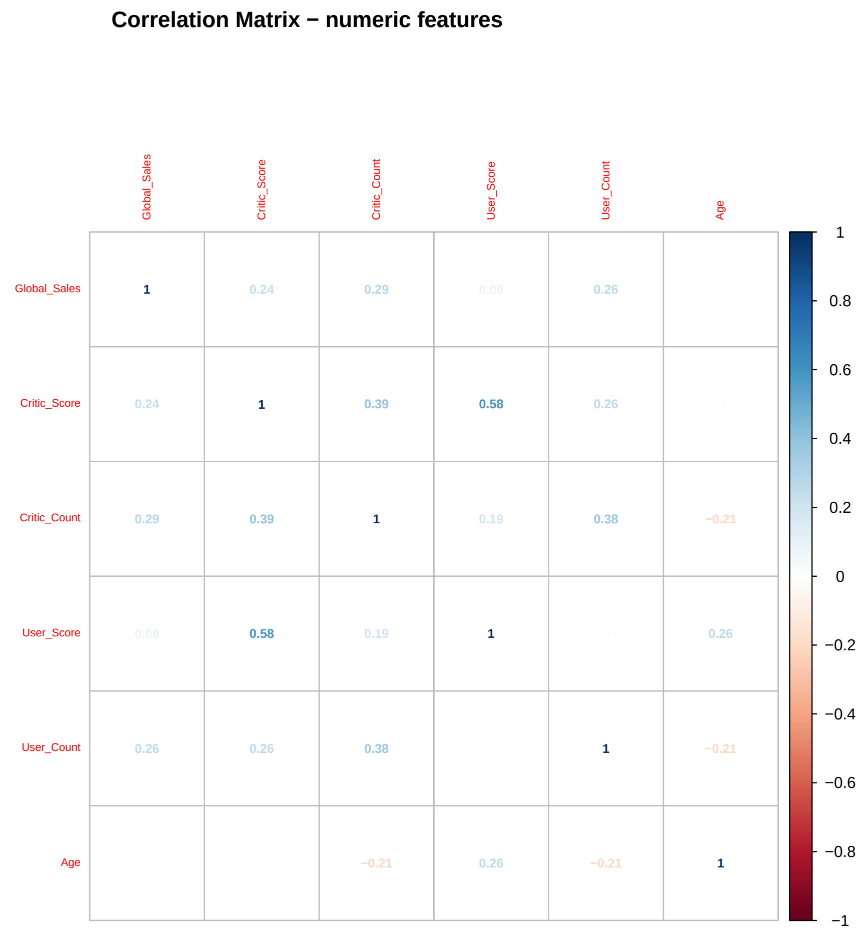
Correlation #1

Since the basic correlation function in R only accepts numeric values, we will first only look at the numeric variables (Global Sales, Age, User Score, User Count, Critic Score, and Critic Count).

```
##                Global_Sales   Critic_Score  Critic_Count     User_Score
## Global_Sales  1.00000000000 0.236336820662  0.2867149652 0.088781482529
## Critic_Score  0.23633682066 1.000000000000  0.3949259241 0.584937694564
## Critic_Count  0.28671496520 0.394925924093  1.0000000000 0.190782093261
## User_Score    0.08878148253 0.584937694564  0.1907820933 1.000000000000
## User_Count    0.26388697590 0.264337350788  0.3753491786 0.007855419303
## Age          -0.00538263803 0.002606199805 -0.2069344634 0.255492747820
##                   User_Count            Age
## Global_Sales  0.263886975898 -0.005382638030
## Critic_Score  0.264337350788  0.002606199805
## Critic_Count  0.375349178555 -0.206934463440
## User_Score    0.007855419303  0.255492747820
## User_Count    1.000000000000 -0.205745055455
## Age          -0.205745055455  1.000000000000
```

```
##                         [,1]
## Global_Sales  1.00000000000
## Critic_Score  0.23633682066
## Critic_Count  0.28671496520
## User_Score    0.08878148253
## User_Count    0.26388697590
## Age          -0.00538263803
```





From these results we find the following:

- None of the numeric features are highly correlated with each other so we won't remove any at this stage.
- Critic Count, User Count and Critic Score have the highest correlation with Global Sales. It's interesting to see that the "Count" variables have a stronger correlation with Global Sales than the "Score" variables. This could be in part due to the fact that "popular" games (ones that receive more hype from critics and users) are more likely to sell more units than those that receive high ratings.
- Age has no correlation with Global Sales.
- User Score has almost zero correlation with Global Sales.

Correlation #2

Since R's correlation function does not permit categorical variables, I will be using a function[18] I found online via Srikanth KS (GitHub: talegari[19]). This function is able to find a correlation matrix for a dataset with mixed variable types. The code is as follows:

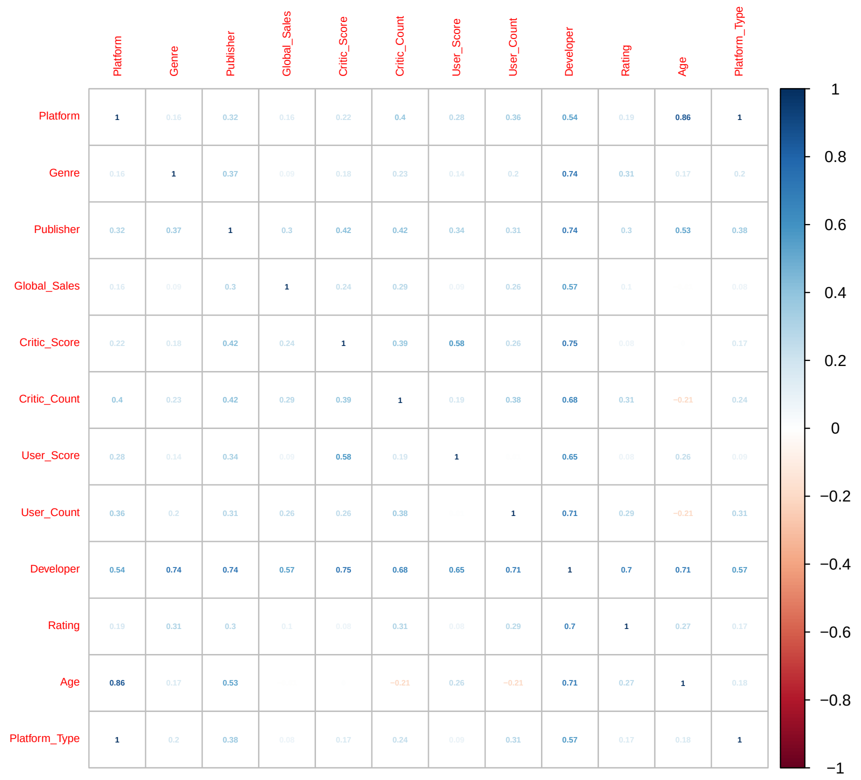
```
cor2 <-function(df){

  stopifnot(inherits(df, "data.frame"))
  stopifnot(sapply(df, class) %in% c("integer"
```

[18]https://gist.github.com/talegari/b514dbbc651c25e2075d88f31d48057b
[19]https://gist.github.com/talegari

```
                                    , "numeric"
                                    , "factor"
                                    , "character"))

  cor_fun <- function(pos_1, pos_2){

    # both are numeric
    if(class(df[[pos_1]]) %in% c("integer", "numeric") &&
       class(df[[pos_2]]) %in% c("integer", "numeric")){
      r <- stats::cor(df[[pos_1]]
                      , df[[pos_2]]
                      , use = "pairwise.complete.obs"
      )
    }

    # one is numeric and other is a factor/character
    if(class(df[[pos_1]]) %in% c("integer", "numeric") &&
       class(df[[pos_2]]) %in% c("factor", "character")){
      r <- sqrt(
        summary(
          stats::lm(df[[pos_1]] ~ as.factor(df[[pos_2]])))[["r.squared"]])
    }

    if(class(df[[pos_2]]) %in% c("integer", "numeric") &&
       class(df[[pos_1]]) %in% c("factor", "character")){
      r <- sqrt(
        summary(
          stats::lm(df[[pos_2]] ~ as.factor(df[[pos_1]])))[["r.squared"]])
    }

    # both are factor/character
    if(class(df[[pos_1]]) %in% c("factor", "character") &&
       class(df[[pos_2]]) %in% c("factor", "character")){
      r <- lsr::cramersV(df[[pos_1]], df[[pos_2]], simulate.p.value = TRUE)
    }

    return(r)
  }

  cor_fun <- Vectorize(cor_fun)

  # now compute corr matrix
  corrmat <- outer(1:ncol(df)
                   , 1:ncol(df)
                   , function(x, y) cor_fun(x, y)
  )

  rownames(corrmat) <- colnames(df)
  colnames(corrmat) <- colnames(df)

  return(corrmat)
}
```

```r
vg_cor <- cor2(vgTrain2)

# Plot 2 - Correlation between numeric and categorical features
set.seed(1234)

include_graphics('./corrplot2.pdf')
```

Correlation Matrix with All Features



We can summarize the results as follows:

- The top 5 features that have the highest correlation with Global Sales are: Developer, Publisher, Critic Count, User Count, and Critic Score.
- Developer is highly correlated with almost every feature in the dataset, most of them above 0.7. It may need to be removed since it is redundant with most of the variables.
- Age and Platform are highly correlated (0.86) – one of them may need to be removed.

2.5.2 Boruta

Boruta is an improvement on the Random Forest – Variable Importance algorithm, another popular feature selection technique. What's special about Boruta is that it accepts variables of different types (numeric, categorical, etc.), works well for classification and regression problems, and considers all features that are significant to the target variable (whereas most other methods only rely on a subset of predictors)[20]. Additionally, the Boruta function can handle relationships between variables. It is also relatively simple to implement and doesn't take too long to run.

```r
### Boruta function
set.seed(1234)

boruta_output <- Boruta(Global_Sales ~ .,
                        data=vgTrain2, doTrace=0)

# Get significant variables including tentatives
boruta_sig <- getSelectedAttributes(boruta_output, withTentative = TRUE)
print(boruta_sig)
```

```
##  [1] "Platform"      "Genre"         "Publisher"     "Critic_Score"
##  [5] "Critic_Count"  "User_Score"    "User_Count"    "Developer"
##  [9] "Rating"        "Age"           "Platform_Type"
```

```r
# Do a tentative rough fix
roughFixMod <- TentativeRoughFix(boruta_output)
boruta_sig <- getSelectedAttributes(roughFixMod)
print(boruta_sig)
```

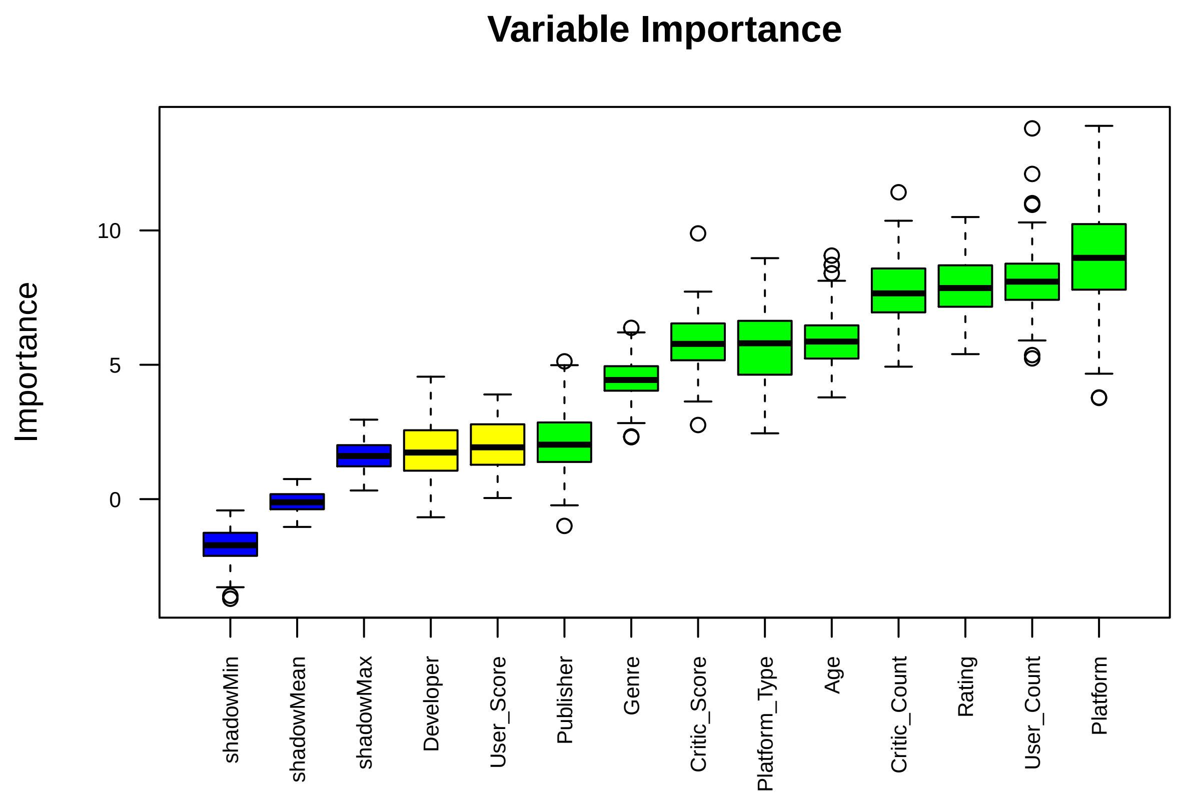
```
##  [1] "Platform"      "Genre"         "Publisher"     "Critic_Score"
##  [5] "Critic_Count"  "User_Score"    "User_Count"    "Developer"
##  [9] "Rating"        "Age"           "Platform_Type"
```

```r
# Variable Importance Scores
imps <- attStats(roughFixMod)
imps2 = imps[imps$decision != 'Rejected', c('meanImp', 'decision')]
head(imps2[order(-imps2$meanImp), ])  # descending sort
```

	meanImp	decision
Platform	8.967114004	Confirmed
User_Count	8.129761490	Confirmed
Rating	7.922186460	Confirmed
Critic_Count	7.720233725	Confirmed
Age	5.877711090	Confirmed
Platform_Type	5.780139947	Confirmed

```r
# Plot variable importance
plot(boruta_output, cex.axis=.7, las=2, xlab="", main="Variable Importance")
```

[20]https://www.datasciencecentral.com/profiles/blogs/select-important-variables-using-boruta-algorithm



The top 5 features according to Boruta are: Platform, User Count, Rating, Critic Count, Age. Developer and User Score performed the worst.

2.5.3 RFE (Recursive Feature Elimination)

Lastly, we will be using Recursive Feature Elimination, a popular wrapper-type feature selection technique. Wrapper-type means that a machine learning algorithm is fed into the method, is wrapped by RFE, and then picks the most relevant features[21]. Filter-based feature selection methods, on the other hand, select features based on the scores the algorithm gives them. According to Jason Brownlee, from Machine Learning Mastery, RFE is technically a "wrapper-style feature selection algorithm that also uses filter-based feature selection internally."[22]

RFE is popular because it is easy to setup and implement and because it selects relevant features effectively. The drawback of the RFE method is that it does not accept features with too many categories (over 53).

We will explore the relevance of all the remaining features in our training set by running a Random Forest algorithm within our RFE function. The code is as follows:

```r
# Recursive Feature Elimination (RFE)

# Convert categorical variables to character
vgTrain2$Developer<-as.character(vgTrain2$Developer)
vgTrain2$Publisher<-as.character(vgTrain2$Publisher)
```

[21]https://machinelearningmastery.com/rfe-feature-selection-in-python/
[22]https://machinelearningmastery.com/rfe-feature-selection-in-python/

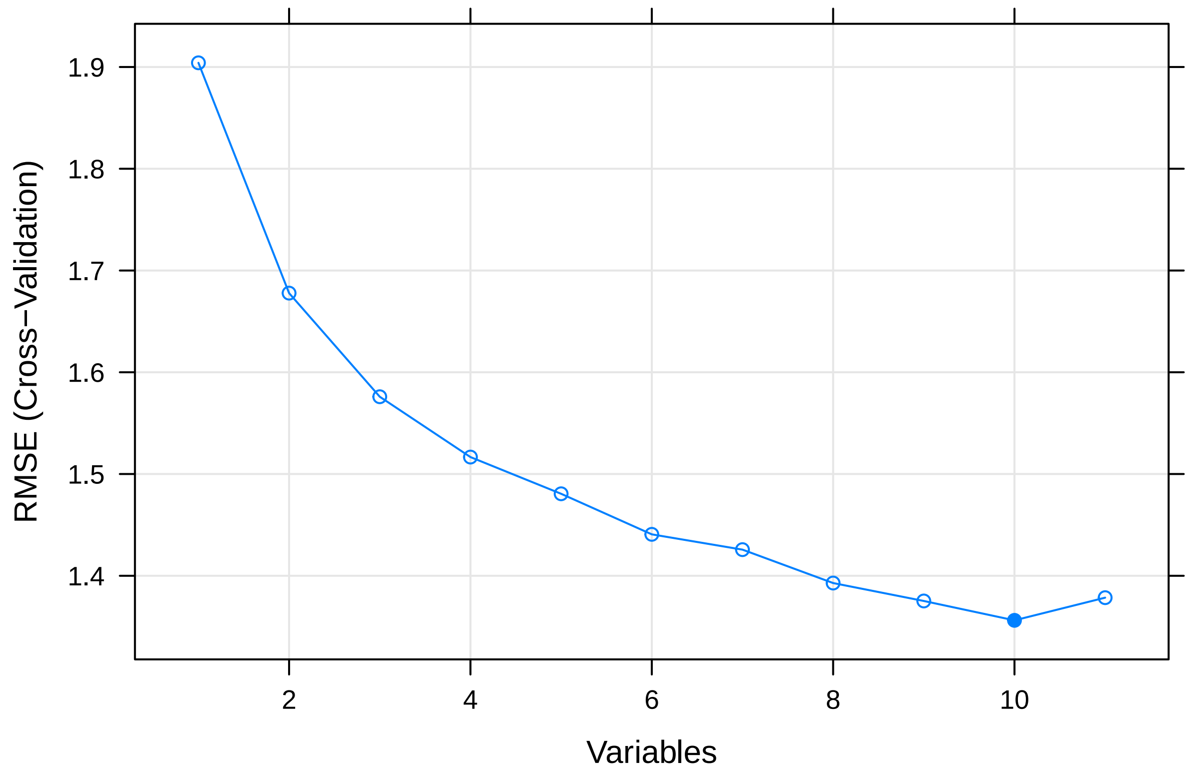
```r
set.seed(1234)

# define the control using a random forest selection function
rfe_ctrl <- rfeControl(functions=rfFuncs, method="cv", number=10)

# run the RFE algorithm
rfe_results <- rfe(vgTrain2[,-4],
               vgTrain2$Global_Sales,
               sizes=c(1:11),
               rfeControl=rfe_ctrl)

# summarize the results
print(rfe_results)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables     RMSE   Rsquared       MAE    RMSESD RsquaredSD      MAESD
##          1 1.904110 0.04798584 0.7555908 0.8544836 0.05241304 0.06975990
##          2 1.677785 0.19868041 0.6314510 0.8646168 0.12983572 0.08411516
##          3 1.575945 0.29933144 0.5774626 0.8841897 0.10205779 0.06366056
##          4 1.516659 0.36782445 0.5435483 0.9013065 0.13451566 0.07905655
##          5 1.480590 0.40890798 0.5340344 0.9179129 0.13860953 0.08175795
##          6 1.440709 0.43356462 0.5177990 0.9198002 0.14637620 0.07909053
##          7 1.425692 0.45166410 0.5115074 0.9294087 0.16285090 0.08201125
##          8 1.392867 0.48063969 0.5024599 0.9093196 0.16388530 0.07438366
##          9 1.375234 0.49143019 0.4968885 0.8944993 0.15625898 0.07022118
##         10 1.356200 0.50906456 0.4973828 0.8975167 0.15101787 0.06757339
##         11 1.378484 0.49913597 0.4981562 0.9053480 0.14910465 0.07064626
##  Selected
##
##
##
##
##
##
##
##
##
##         *
##
##
## The top 5 variables (out of 10):
##    User_Count, Platform, Critic_Count, Critic_Score, Age
```

```r
# list the chosen features
predictors(rfe_results)
```

```
##  [1] "User_Count"    "Platform"      "Critic_Count"  "Critic_Score"
##  [5] "Age"            "Publisher"      "Platform_Type" "Genre"
##  [9] "Rating"         "User_Score"
```

```
# plot the results
plot(rfe_results, type=c("g", "o"))
```



The results are as follows:

- The top 5 features according to RFE are: User_Count, Platform, Critic_Count, Critic_Score, Age.
- The optimal number of features that produced the lowest RMSE was 10, although 8 features produce comparable results.

2.5.4 Final Results

Since we have used 4 different feature selection methods (2 correlation techniques, Boruta, and RFE), we want to look at all their results together to decide which features to keep for modeling and which features to remove.

```
## # A tibble: 5 x 4
##   Corr1       Corr2       Boruta      RFE
##   <chr>       <chr>       <chr>       <chr>
## 1 CriticCount Developer   Platform    UserCount
## 2 UserCount   Publisher   UserCount   Platform
## 3 CriticScore CriticCount Rating      CriticCount
## 4 UserScore   UserCount   CriticCount CriticScore
## 5 Age         CriticScore Age         Age
```

From this table, we make the following observations:

- Critic Count and User Count appeared in every top 5 ranking.
- Critic Score and Age were featured 3 times.
- Platform was featured twice.
- User Score, Publisher, Developer, and Rating were only featured once.
- Genre and Platform Type were not featured at all.

Features to keep: Critic Count, User Count, Critic Score, Age, and Platform.

Features to remove: User Score, Publisher, Developer, Rating, Genre, Platform Type.

2.6 Data Preprocessing for Modeling

These are the steps we will carry out to prepare the training set for modeling:

1. Remove the features we won't be using for modeling, based on the feature selection methods we implemented.
2. Check if there are any missing values – we will use the training set with no missing values since data imputation of such a large number of observations (over 50%) could greatly skew our results. Furthermore, the variables with the highest number of missing values are also the ones with the strongest relationship with the target outcome (eg. User Count, Critic Count, Critic Score) which may create a bias. We end up losing over 50% of the data but for now this seems to be the best option.
3. Center and scale the numeric features – some models do not perform well on unscaled data.
4. Create dummy variables for categorical variables in training set since some algorithms like linear regression only accept numeric variables.

The following code implements on all these steps:

```
# Data Preprocessing for Modeling ####

# Select subset of features for modeling (based on Feature Selection)
vgTrain2<-vgTrain2%>%
  dplyr::select(-c(User_Score,Publisher,Developer,Rating,Genre,Platform_Type))
colnames(vgTrain2) #check column names
```

```
## [1] "Platform"     "Global_Sales" "Critic_Score" "Critic_Count" "User_Count"
## [6] "Age"
```

```
nrow(vgTrain2) #check number of observations
```

```
## [1] 5475
```

```
train_set<-vgTrain2   #rename to train set

# Check if any NAs in datasets
apply(vgTrain2,2,function(x){sum(is.na(x))})
```

```
##     Platform Global_Sales Critic_Score Critic_Count   User_Count          Age
##            0            0            0            0            0            0
```

```r
# Center and scale numeric features

preProcValues <- preProcess(train_set, method = c("center", "scale"))
train_set <- predict(preProcValues, train_set)

# Dummify categorical variables in train set

t_dummy <- dummyVars(" ~ .", data = train_set)
train_dummy <- data.frame(predict(t_dummy, newdata = train_set))
head(train_dummy)
```

	Platform.2600	Platform.3DO	Platform.3DS	Platform.DC	Platform.DS	Platform.GB	Platform.GBA	Plat
1	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
5	0	0	0	0	1	0	0	
6	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	
8	0	0	0	0	1	0	0	

Our training set is now ready for modeling.

3 Modeling Methods

Since the outcome that we are trying to predict is continuous, in this case Global Sales, we will be training four different models: Linear Regression, Elastic Net, Support Vector Machine, and Random Forest.

3.1 Linear Regression

We start with the simplest of our models – linear regression. Regression is a type of predictive modeling algorithm that focuses on "the relationship between a dependent (target) variable and an independent variable(s) (predictors)", under the assumption that the independent variable(s) cause(s) the dependent variable [23]. In other words, the independent variables (predictors) are used to predict the outcome (target variable).

The benefit of regression is that it allows us to analyze the varying levels of impact of multiple different features on the target variable. It also accepts different types of data such as nominal, interval, or categorical variables for analysis [24].

Linear regression, which we will be using as our first model, makes a few assumptions about the data:

- There is a linear relationship between the predictors and the target variable.
- The variables have a Gaussian distribution.
- The variables are not correlated with each other [25].

The model operates by choosing coefficients for each independent predictor that minimizes a loss function (such as RMSE) [26]. However, if the coefficients are large, it can lead to overfitting on the training dataset, which means the model will not generalize well to new data. In order to counteract this pitfall, we adopt an approach called regularization, which penalizes large coefficients.

[23]https://medium.com/swlh/predictive-modelling-using-linear-regression-e0e399dc4745#:~:text=Linear%20regression%20is%20one%20of,given%

[24]https://medium.com/swlh/predictive-modelling-using-linear-regression-e0e399dc4745#:~:text=Linear%20regression%20is%20one%20of,given%

[25]https://www.pluralsight.com/guides/linear-lasso-and-ridge-regression-with-r

[26]https://www.pluralsight.com/guides/linear-lasso-and-ridge-regression-with-r

Linear Regression algorithm require inputs to be numeric so we will create dummy variables for the categorical features.

3.2 Elastic Net Regression

Elastic Net Regression is a regularization algorithm that combines the properties of Lasso and Ridge Regression (L1 and L2), two other penalty functions.

The L1 penalty function penalizes a model based on the “sum of the absolute coefficient values” and the L2 penalty penalizes based on the “sum of the squared coefficient values” [27]. Both functions minimize the size of all coefficients, however L1 allows some coefficients to be zero, thus removing the predictor from the model, whereas L2 does not allow any predictors to be removed from the model [28].

Elastic Net is a type of penalized linear regression that combines both of these penalties during the training stage. The model has two hyperparameters: “alpha” and “lambda”. Alpha is used to assign weights to each of the L1 and L2 penalties. The lambda value controls how much the sum of both penalties impacts the loss function [29].

Through these hyperparameters, elastic net allows a balance between the two penalties, which can help to build a better performing model.

3.3 Support Vector Machine

Support Vector Machine (SVM) is a powerful machine learning algorithm, popularized in the 1990s and still widely used today. The algorithm can be applied to both regression and classification problems. Although the model is linear in nature, it can solve non-linear and often very complex problems with minimal tuning [30]. The goal of SVM is to create a line or hyperplane that best separates the data inputs (features) into classes [31]. SVM works particularly well with small to medium sized datasets like the videogames dataset, as the model can take a while to run.

We will be using Support Vector Regression with RBF (Radial Basis Function Kernel) – this is a default kernel that is recommended when there might be a non-linear relationship between features and when it is tricky to select a particular kernel. According to this explanation[32], “the kernel function transforms our data from non-linear space to linear space.” This permits the algorithm to find the best fit and then data is plotted accordingly. We are adopting this method because it is quite flexible, especially when dealing with different variable distributions. The risk of over-fitting is less with SVM compared to other models as they generalize better to unseen data [33].

SVM only permits numeric inputs so we will convert our categorial features to dummy variables.

3.4 Random Forest

Lastly, we will be implementing another popular machine learning algorithm, Random Forest, which can be applied to both regression and classification problems. Random Forests improve on decision tree models by “averaging multiple decision trees (a forest of trees constructed with randomness)” [34]. It does this by running many random trees (through a method called bootstrap aggregation or bagging) in order to generate

[27]https://machinelearningmastery.com/elastic-net-regression-in-python/
[28]https://machinelearningmastery.com/elastic-net-regression-in-python/
[29]https://machinelearningmastery.com/elastic-net-regression-in-python/
[30]https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989
[31]https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989
[32]https://rpubs.com/linkonabe/SLSvsSVR
[33]https://statinfer.com/204-6-8-svm-advantages-disadvantages-applications/
[34]https://rafalab.github.io/dsbook/examples-of-algorithms.html

several predictions which are then averaged out to produce a final prediction [35]. Thus, Random Forests often produce more accurate predictions than other ML models.

Random Forests are robust algorithms that can work with any type of data. They accept data with different distributions and can handle collinearity between variables. They are also less susceptible to outliers than other models. This flexibility, and the fact that they are relatively easy to implement, make them a very attractive choice for our prediction problem. Implementing Random Forest will also help us understand the importance of different features in predicting the target outcome (Global Sales) since RF is a type of feature selection technique.

One of the drawbacks of Random Forests is that their results may be hard to interpret. One way around this is to look at Variable Importance which aggregates how often a particular feature was used in the individual trees [36].

4 Results

4.1 Model Evaluation Function

We start off by defining our loss function, Root Mean Squared Error (RMSE), which we explained in Section 1.3.

```
# Define Root Mean Squared Error (RMSE) - loss function
RMSE <- function(true_sales, predicted_sales){
  sqrt(mean((true_sales - predicted_sales)^2))
}
```

4.2 Control Parameters

Before we can build our models, we have to set control parameters which will be implemented when we train the models on our dataset.

For the purposes of this project, we will be tuning the "Method" parameter which is used to evaluate the accuracy of our models. In other words, how well our model generalizes to unseen data. There are several methods to do this, including splitting the dataset into train/validation/test sets, but because we don't have a large dataset to work with, we have chosen Repeated K-Fold Cross Validation which builds on k-fold cross validation.

K-fold cross validation is a robust technique that splits the data into k-subsets and for each iteration, the model holds back a subset (as a validation set) and trains on all the other data. The overall accuracy is then calculated by aggregating the accuracies produced by each iteration. Repeated CV repeats this process several times and calculates the model's accuracy by taking the mean of the repeats.

We will set the number of k-folds at 10, the number of repeats at 3, and save only the final predictions. We will also highlight that the metric of interest is RMSE, which will be input into the training model. The following code achieves these steps:

```
# Set control parameters - run algorithms using 10-fold cross validation repeated 3 times

trControl <- trainControl(method = "repeatedcv",
                          number = 10,
                          repeats=3,
```

[35] https://rafalab.github.io/dsbook/examples-of-algorithms.html
[36] https://rafalab.github.io/dsbook/examples-of-algorithms.html

```
                           savePredictions="final")

metric <- "RMSE"
```

4.3 Linear Regression

We run the linear regression using the train set that has been centered, scaled, and with dummies to represent categorical variables. The code to run the model is as follows:

```
# Linear Regression Model
set.seed(1234)

fit_lr<-train(Global_Sales~.,
              data=train_dummy,
              method="lm",
              metric=metric,
              trControl=trControl)

# Predict on training set
train_lr<-predict(fit_lr,train_dummy)

# Training error
RMSE(train_set$Global_Sales, train_lr)
```

```
## [1] 0.9088345706
```

```
# Validation error
rmse_lr<-fit_lr$results['RMSE']
rmse_lr
```

RMSE
0.833553442

We predict on the training set in order to obtain a training error. We retrieve the validation error from the train object. It is difficult to decipher whether this is a good fit because the validation error is lower than the training error, which is usually the other way around. It's possible that our training set had many 'hard' cases to learn or that our validation set had many 'easy' cases to predict.

```
# Check variable importance
varImp(fit_lr)
```

```
## lm variable importance
##
##                     Overall
## User_Count      100.0000000
## Critic_Count     73.1744638
## Critic_Score     66.6744205
## Platform.PC      39.6840984
## Age              22.6034642
## Platform.Wii     22.3014474
## Platform.XB      19.8863095
## Platform.GC      14.5023520
## Platform.DC    11.2512248
## Platform.PSV   10.6472231
## Platform.PSP    9.0271634
## Platform.PS     8.4916969
## Platform.X360   7.4270825
## Platform.GBA    6.6477899
## Platform.PS4    5.9314092
## Platform.PS2    4.5085473
## Platform.WiiU   2.9275819
## Platform.PS3    1.3565171
## Platform.3DS    0.5037908
## Platform.DS     0.0000000
```

According to the LR model's Variable Importance, User Count, Critic Count, and Critic Score are the most important features.

We will create a table to store the RMSE results (Training Error and Validation Error) from each model so we can compare and select the best performing one.

```
# Create a table of results to compare models' performance
rmse_results <- tibble(Method = "Linear Regression",
                       "Training Error" = RMSE(train_set$Global_Sales, train_lr),
                       "Validation Error" = rmse_lr$RMSE)

knitr::kable(rmse_results) %>% kable_styling()
```

Method	Training Error	Validation Error
Linear Regression	0.9088346	0.8335534

4.4 Elastic Net

Next, we run the Elastic Net Regression model as follows:

```
# -> Elastic Net Regression ####
set.seed(1234)

fit_glmnet<-train(Global_Sales~.,
                  data=train_dummy,
                  method="glmnet",
                  metric=metric,
                  trControl=trControl,
                  tuneLength=10)

# Predict on training set
train_glmnet<-predict(fit_glmnet,train_dummy)
```

The train function allows us to randomly search for the optimal parameters (in this case lambda and alpha) through the tuneLength argument, which sets the total number of unique combinations for the model to search.

```
# Training error
RMSE(train_set$Global_Sales, train_glmnet)
```

```
## [1] 0.9094815423
```

```
# Validation error
rmse_glmnet<-fit_glmnet$results[row.names(fit_glmnet$bestTune),'RMSE']
rmse_glmnet
```

```
## [1] 0.8330187108
```

Method	Training Error	Validation Error
Linear Regression	0.9088346	0.8335534
Elastic Net	0.9094815	0.8330187

Elastic Net produced a very small improvement on the RMSEs from linear regression. Perhaps the model needs better tuning or that the coefficients didn't need much regularization.

```
# Check variable importance
varImp(fit_glmnet)
```

```
## glmnet variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                  Overall
## Platform.PC   100.000000
## Platform.Wii   98.833118
## Platform.PS    77.631704
## Platform.DC    59.216903
## User_Count     47.138325
## Platform.XB    43.498984
## Critic_Count   39.430993
## Critic_Score   31.765375
## Platform.GC    25.444895
## Platform.DS    25.360490
## Platform.PSV   23.900421
## Platform.3DS   18.269939
## Age            13.795034
## Platform.PS3   13.092601
## Platform.XOne  10.054685
## Platform.PS2    8.816451
## Platform.PSP    5.702831
## Platform.PS4    0.203719
## Platform.3DO    0.000000
## Platform.SAT    0.000000
```

The most important features, according to Elastic Net, are the Platform categories.

4.5 Support Vector Machine

As mentioned earlier, we will be using a type of SVM called Support Vector Regression with RBF (Radial Basis Function Kernel) which operates with a default kernel.

```
# -> Support Vector Machine ####
set.seed(1234)

fit_svm<-train(Global_Sales~.,
               data=train_dummy,
               method="svmRadial",
               metric=metric,
               trControl=trControl)

# Predict on training set
train_svm<-predict(fit_svm,train_dummy)

# Training error
RMSE(train_set$Global_Sales, train_svm)
```

```
## [1] 0.8618670283
```

```
# test_set error
rmse_svm<-fit_svm$results[row.names(fit_svm$bestTune),'RMSE']
rmse_svm
```

```
## [1] 0.7910237889
```

Method	Training Error	Validation Error
Linear Regression	0.9088346	0.8335534
Elastic Net	0.9094815	0.8330187
SVM	0.8618670	0.7910238

SVM produced a significant improvement on the training error and validation error. It seems to have performed with greater accuracy than linear regression or elastic net.

```
# Check variable importance
varImp(fit_svm)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                   Overall
## Critic_Count  1.000000e+02
## User_Count    8.470972e+01
## Critic_Score  6.794493e+01
## Platform.Wii  8.457399e+00
## Platform.PC   7.070907e+00
## Platform.XB   4.185665e+00
## Platform.PS   3.408021e+00
## Platform.GC   1.870106e+00
## Platform.PS3  1.713750e+00
## Platform.X360 1.568693e+00
## Platform.PSP  1.337465e+00
## Platform.PSV  1.138787e+00
## Platform.PS4  7.530620e-01
## Platform.GBA  4.093848e-01
## Platform.DC   1.177959e-01
## Platform.PS2  6.218516e-02
## Platform.3DS  4.076212e-02
## Age           3.279019e-02
## Platform.XOne 2.174097e-02
## Platform.DS   1.520683e-04
```

Similar to linear regression, the most important features according to SVM are Critic Count, User Count and Critic Score.

4.6 Random Forest

We will be training two Random Forest models – one with default parameters and the other with tuned parameters. We will start with the default option.

Model 1

```r
# Rf Model 1 - Run the model with default parameters
set.seed(1234)

fit_rf1 <- train(Global_Sales~.,
                 data = train_dummy,
                 method = "rf",
                 metric=metric,
                 trControl = trControl)

fit_rf1
```

```
## Random Forest
##
## 5475 samples
##   35 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 4927, 4928, 4929, 4928, 4928, 4927, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE          Rsquared      MAE
##    2    0.8751331146  0.3036296796  0.3551320520
##   18    0.7726071920  0.3640580321  0.2695335421
##   35    0.7998688512  0.3475589402  0.2746174785
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 18.
```

```
# Predict on training set
train_rf1<-predict(fit_rf1,train_dummy)

# Training error
RMSE(train_set$Global_Sales, train_rf1)
```

```
## [1] 0.4640654142
```

```
# Validation error
rmse_rf1<-fit_rf1$results[row.names(fit_rf1$bestTune),'RMSE']
rmse_rf1
```

```
## [1] 0.772607192
```

Method	Training Error	Validation Error
Linear Regression	0.9088346	0.8335534
Elastic Net	0.9094815	0.8330187
SVM	0.8618670	0.7910238
RandomForest 1	0.4640654	0.7726072

We see an improvement in both training error and validation error compared to the other models, however, the training error is significantly lower than the validation error. This could be a sign of the model overfitting which means it might not generalize well to unseen data (the test set). This makes sense as we have not tuned the parameters.

Model 2

We will fine tune the parameters which usually have the greatest impact on Random Forest's performance – mtry, maxnodes, and ntrees. According to the caret package vignette[37], these parameters are defined as follows:

- mtry: number of randomly selected predictors
- ntrees: number of trees to train
- maxnodes: max number of nodes the forest can have

We find the best mtry using the following code:

```
set.seed(1234)

# Search for best mtry
tuneGrid <- expand.grid(.mtry = c(1: 10)) #Construct a vector with values from 3:10

rf_mtry <- train(Global_Sales~.,
                 data = train_dummy,
                 method = "rf",
                 tuneGrid = tuneGrid,
                 trControl = trControl,
                 importance = TRUE,
                 nodesize = 14,
                 ntree = 300)
print(rf_mtry)
```

[37]https://topepo.github.io/caret/train-models-by-tag.html#random-forest

```
## Random Forest
##
## 5475 samples
##   35 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 4927, 4928, 4929, 4928, 4928, 4927, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE          Rsquared      MAE
##    1    0.9123049899  0.2523802732  0.3825044101
##    2    0.8742656266  0.2988931215  0.3544100546
##    3    0.8308107096  0.3353555919  0.3246163679
##    4    0.7965513639  0.3559975074  0.3003125599
##    5    0.7762743270  0.3659211471  0.2852894982
##    6    0.7643487516  0.3769407394  0.2766337214
##    7    0.7585641122  0.3823116976  0.2716119867
##    8    0.7580022527  0.3800262691  0.2695846660
##    9    0.7570585016  0.3801991253  0.2683229807
##   10    0.7555432717  0.3826532087  0.2675385705
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 10.
```

```
rf_mtry$bestTune$mtry # best value of mtry
```

```
## [1] 10
```

```
best_mtry <- rf_mtry$bestTune$mtry #store best mtry for final model
best_mtry
```

```
## [1] 10
```

The best mtry is 10 which we will save to use in our model. Next we find the best values for maxnodes and ntrees, using a helpful function from Guru99.com[38].

```
set.seed(1234)

store_maxnode <- list() #The results of the model will be stored in this list
tuneGrid <- expand.grid(.mtry = best_mtry) #Use the best value of mtry
for (maxnodes in c(10: 25)) {   #Compute the model with different values of maxnodes from 10:25.
  set.seed(1234)
  rf_maxnode <- train(Global_Sales~.,
                      data = train_dummy,
                      method = "rf",
                      tuneGrid = tuneGrid,
                      trControl = trControl,
                      importance = TRUE,
                      nodesize = 14,
```

[38]https://www.guru99.com/r-random-forest-tutorial.html

```
                       maxnodes = maxnodes,
                       ntree = 300)
  current_iteration <- toString(maxnodes)   #Store as a string variable the value of maxnode.
  store_maxnode[[current_iteration]] <- rf_maxnode   #Save the result of the model in the list.
}
results_maxnode <- resamples(store_maxnode)  #Arrange the results of the model
summary(results_maxnode)  #summarize results
```

```
##
## Call:
## summary.resamples(object = results_maxnode)
##
## Models: 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25
## Number of resamples: 30
##
## MAE
##         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 10 0.2661351 0.2960269 0.3047855 0.3086186 0.3173058 0.3953916    0
## 11 0.2610748 0.2921644 0.3042775 0.3061397 0.3162970 0.3930979    0
## 12 0.2615137 0.2909341 0.3031994 0.3046408 0.3127561 0.3911805    0
## 13 0.2614877 0.2886005 0.3006950 0.3032499 0.3126227 0.3927630    0
## 14 0.2556214 0.2873177 0.2999168 0.3013371 0.3118947 0.3902010    0
## 15 0.2523301 0.2854715 0.2980072 0.2995029 0.3077639 0.3872026    0
## 16 0.2517932 0.2831671 0.2958612 0.2987594 0.3087881 0.3865879    0
## 17 0.2516090 0.2837672 0.2958061 0.2977017 0.3062429 0.3861265    0
## 18 0.2496228 0.2813435 0.2931864 0.2960678 0.3046678 0.3854768    0
## 19 0.2495803 0.2799935 0.2946559 0.2952915 0.3055154 0.3844024    0
## 20 0.2471554 0.2773380 0.2928889 0.2941740 0.3052312 0.3818914    0
## 21 0.2469700 0.2778147 0.2914776 0.2935578 0.3031091 0.3831359    0
## 22 0.2444958 0.2776970 0.2921557 0.2927694 0.3031966 0.3802669    0
## 23 0.2473905 0.2754217 0.2879695 0.2916179 0.3016317 0.3812724    0
## 24 0.2439289 0.2748116 0.2892512 0.2914281 0.3007534 0.3809243    0
## 25 0.2442385 0.2732169 0.2884293 0.2900519 0.3011597 0.3787679    0
##
## RMSE
##         Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## 10 0.4280166 0.5935048 0.7129978 0.8098176 0.8447720 1.939716    0
## 11 0.4192354 0.5930043 0.7111947 0.8063342 0.8395595 1.930832    0
## 12 0.4263158 0.5872525 0.7086255 0.8044039 0.8342749 1.932660    0
## 13 0.4266490 0.5884381 0.7077845 0.8022512 0.8285496 1.928769    0
## 14 0.4166734 0.5851742 0.7049650 0.7997562 0.8293844 1.929764    0
## 15 0.4107967 0.5823199 0.7035273 0.7973120 0.8242493 1.929293    0
## 16 0.4109545 0.5839732 0.6982770 0.7958858 0.8257822 1.919091    0
## 17 0.4128316 0.5752926 0.7018161 0.7946863 0.8226881 1.920420    0
## 18 0.4085431 0.5846703 0.6955928 0.7934737 0.8137777 1.924573    0
## 19 0.4103823 0.5805922 0.6960873 0.7919705 0.8162815 1.918671    0
## 20 0.4067161 0.5771805 0.6958446 0.7893657 0.8103121 1.914256    0
## 21 0.4083588 0.5788817 0.6916199 0.7895607 0.8092049 1.919088    0
## 22 0.4046762 0.5765562 0.6893875 0.7874433 0.8092613 1.906577    0
## 23 0.4131223 0.5750699 0.6936914 0.7872349 0.8062443 1.916837    0
## 24 0.4045269 0.5766006 0.6884812 0.7848640 0.8034525 1.910890    0
## 25 0.4081116 0.5742572 0.6891040 0.7848696 0.8036476 1.913772    0
##
```

```
## Rsquared
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 10 0.05728716 0.2351981 0.3051158 0.3038010 0.3987942 0.5181824    0
## 11 0.05596758 0.2425157 0.3139106 0.3101635 0.4025136 0.5377060    0
## 12 0.06637931 0.2473040 0.3093498 0.3113815 0.4019333 0.5139225    0
## 13 0.07439419 0.2501694 0.3198877 0.3129324 0.4130746 0.5077576    0
## 14 0.07147375 0.2458258 0.3171455 0.3166189 0.4140326 0.5388643    0
## 15 0.07316516 0.2513528 0.3268972 0.3226544 0.4083193 0.5516353    0
## 16 0.06799626 0.2481036 0.3338108 0.3231923 0.4108869 0.5505811    0
## 17 0.07569482 0.2574964 0.3207698 0.3244858 0.4129054 0.5438671    0
## 18 0.07829372 0.2581432 0.3253792 0.3252230 0.4199040 0.5526325    0
## 19 0.07930096 0.2611222 0.3341250 0.3285915 0.4158823 0.5406759    0
## 20 0.07927246 0.2677522 0.3298382 0.3323752 0.4218525 0.5510110    0
## 21 0.09203374 0.2634410 0.3284039 0.3311685 0.4215977 0.5459785    0
## 22 0.10903352 0.2647757 0.3277118 0.3344037 0.4276377 0.5563752    0
## 23 0.07731678 0.2563119 0.3365227 0.3349562 0.4293454 0.5426527    0
## 24 0.07282148 0.2704105 0.3369880 0.3378437 0.4273906 0.5538724    0
## 25 0.08604551 0.2674406 0.3321609 0.3382553 0.4338939 0.5419018    0
```

```
# Search for best ntree - same methodology as maxnode
set.seed(1234)

store_maxtrees <- list()
for (ntree in c(250, 300, 350, 400, 450, 500, 550, 600, 800, 1000, 2000)) {
  set.seed(1234)
  rf_maxtrees <- train(Global_Sales~.,
                       data = train_dummy,
                       method = "rf",
                       tuneGrid = tuneGrid,
                       trControl = trControl,
                       importance = TRUE,
                       nodesize = 14,
                       maxnodes = 24,
                       ntree = ntree)
  key <- toString(ntree)
  store_maxtrees[[key]] <- rf_maxtrees
}
results_tree <- resamples(store_maxtrees)
summary(results_tree)
```

```
##
## Call:
## summary.resamples(object = results_tree)
##
## Models: 250, 300, 350, 400, 450, 500, 550, 600, 800, 1000, 2000
## Number of resamples: 30
##
## MAE
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 250  0.2432869 0.2742588 0.2893359 0.2913399 0.3013516 0.3807115    0
## 300  0.2439289 0.2748116 0.2892512 0.2914281 0.3007534 0.3809243    0
## 350  0.2442759 0.2737524 0.2892557 0.2912478 0.3006557 0.3810298    0
## 400  0.2439405 0.2743610 0.2893282 0.2913290 0.3009716 0.3804772    0
## 450  0.2434720 0.2743114 0.2889657 0.2912462 0.3010045 0.3797215    0
## 500  0.2433349 0.2743034 0.2893117 0.2911199 0.3011973 0.3799668    0
## 550  0.2431463 0.2741975 0.2893626 0.2911223 0.3009945 0.3799551    0
## 600  0.2438239 0.2743527 0.2889750 0.2910483 0.3006621 0.3794678    0
## 800  0.2437944 0.2748203 0.2893836 0.2910323 0.3007964 0.3790006    0
## 1000 0.2436444 0.2749339 0.2896635 0.2910971 0.3010193 0.3788512    0
## 2000 0.2436487 0.2751244 0.2893856 0.2910736 0.3017616 0.3791260    0
##
## RMSE
##           Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## 250  0.4036674 0.5783058 0.6890898 0.7847546 0.8034516 1.909782    0
## 300  0.4045269 0.5766006 0.6884812 0.7848640 0.8034525 1.910890    0
## 350  0.4058500 0.5772543 0.6885447 0.7845800 0.8021380 1.912422    0
## 400  0.4051348 0.5788274 0.6882471 0.7849264 0.8034613 1.912258    0
## 450  0.4032994 0.5791715 0.6885159 0.7848602 0.8032260 1.912618    0
## 500  0.4036194 0.5786214 0.6878600 0.7847677 0.8036866 1.911726    0
## 550  0.4032162 0.5789247 0.6882760 0.7847622 0.8034701 1.912423    0
## 600  0.4050524 0.5787129 0.6886855 0.7847341 0.8023953 1.911154    0
## 800  0.4043826 0.5801849 0.6881699 0.7848444 0.8010781 1.911914    0
## 1000 0.4052862 0.5795210 0.6891938 0.7853162 0.8040426 1.912732    0
## 2000 0.4050307 0.5773580 0.6897725 0.7856434 0.8043644 1.913816    0
##
## Rsquared
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 250  0.07252837 0.2698595 0.3398290 0.3378349 0.4251462 0.5563563    0
## 300  0.07282148 0.2704105 0.3369880 0.3378437 0.4273906 0.5538724    0
## 350  0.07445831 0.2703554 0.3453226 0.3382949 0.4277834 0.5531613    0
## 400  0.07454173 0.2702038 0.3406619 0.3378477 0.4299826 0.5541747    0
## 450  0.07631070 0.2689265 0.3407340 0.3380624 0.4309667 0.5578109    0
## 500  0.07622538 0.2701140 0.3396155 0.3382501 0.4317875 0.5559478    0
## 550  0.07732525 0.2672578 0.3384742 0.3381917 0.4311805 0.5567067    0
## 600  0.07474124 0.2672851 0.3399856 0.3383993 0.4312137 0.5510384    0
## 800  0.07574861 0.2685553 0.3443273 0.3383532 0.4279415 0.5532246    0
## 1000 0.07743896 0.2667601 0.3418747 0.3373435 0.4252267 0.5497107    0
## 2000 0.07969545 0.2654852 0.3361395 0.3371526 0.4277376 0.5507298    0
```

By assessing the results of both functions, we find that the best maxnode=24 and the best ntree=600. We're now going to train our model with these tuned parameters as follows:

```r
# Train new model with tuned parameteres
set.seed(1234)

fit_rf2 <- train(Global_Sales~.,
                 train_dummy,
                 method = "rf",
                 tuneGrid = tuneGrid,    # best mtry
                 trControl = trControl,
                 importance = TRUE,
                 nodesize = 14,
                 ntree = 600,    # best ntree
                 maxnodes = 24,
                 metric=metric)  # best maxnodes

# Predict on training set
train_rf2<-predict(fit_rf2,train_dummy)
```

```
# Training error
RMSE(train_set$Global_Sales, train_rf2)
```

```
## [1] 0.7668527
```

```
# Validation error
rmse_rf2<-fit_rf2$results[row.names(fit_rf2$bestTune),'RMSE']
rmse_rf2
```

```
## [1] 0.7847341
```

```
# Check variable importance
varImp(fit_rf2)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                Overall
## Platform.DS   100.000
## Platform.Wii   91.167
## Platform.PS    79.342
## Platform.PC    72.341
## Critic_Count   67.715
## Platform.PS2   63.008
## Platform.PSV   51.792
## Critic_Score   45.169
## Platform.XB    44.942
## User_Count     44.662
## Age            38.449
## Platform.3DS   34.004
## Platform.XOne  32.834
## Platform.X360  27.891
## Platform.GBA   27.831
## Platform.GC    23.780
## Platform.PS4   20.725
## Platform.DC    20.342
## Platform.WiiU  19.734
## Platform.PS3    5.168
```

From the results we can see that both the training error and validation error are higher than those produced by the first Random Forest model. However, as we mentioned previously the very low training error could have been an indication of the model overfitting. In the case of this model, the gap between the two errors is more optimal.

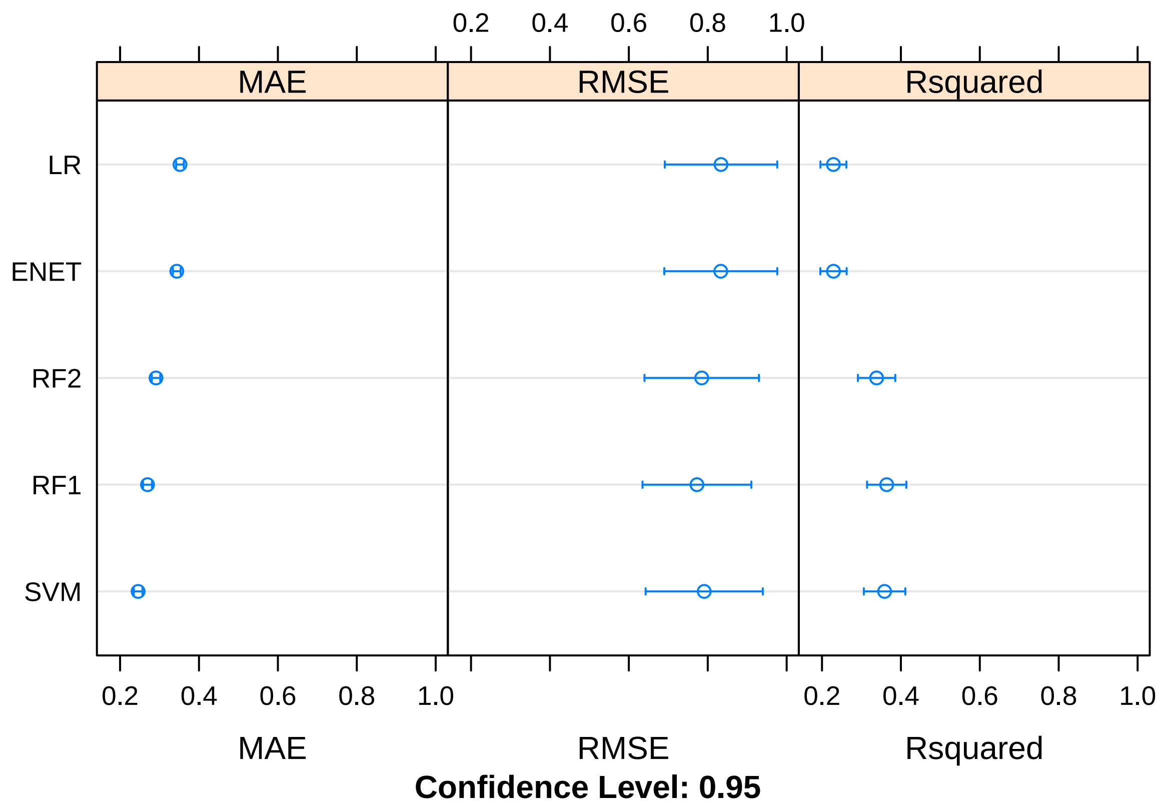
In terms of Variable Importance, Random Forest ranked the Platform categories and Critic Count at the top.

4.7 Final Validation

Compare Models

Method	Training Error	Validation Error
Linear Regression	0.9088346	0.8335534
Elastic Net	0.9094815	0.8330187
SVM	0.8618670	0.7910238
RandomForest 1	0.4640654	0.7726072
RandomForest 2	0.7668527	0.7847341

```
##
## Call:
## summary.resamples(object = results)
##
## Models: LR, ENET, SVM, RF1, RF2
## Number of resamples: 30
##
## MAE
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LR   0.3135072 0.3328614 0.3484084 0.3518139 0.3591478 0.4257314    0
## ENET 0.3025448 0.3258144 0.3400652 0.3437371 0.3519115 0.4187653    0
## SVM  0.2040151 0.2285452 0.2442224 0.2457058 0.2509489 0.3362363    0
## RF1  0.2181146 0.2530165 0.2647777 0.2695335 0.2820246 0.3554812    0
## RF2  0.2438239 0.2743527 0.2889750 0.2910483 0.3006621 0.3794678    0
##
## RMSE
##           Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## LR   0.4829321 0.6350715 0.7346104 0.8335534 0.8679719 1.949187    0
## ENET 0.4741029 0.6361252 0.7332945 0.8330187 0.8720187 1.953663    0
## SVM  0.4087636 0.5525168 0.6947137 0.7910238 0.8321862 1.930422    0
## RF1  0.4252173 0.5721168 0.6532322 0.7726072 0.7955795 1.856141    0
## RF2  0.4050524 0.5787129 0.6886855 0.7847341 0.8023953 1.911154    0
##
## Rsquared
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LR   0.06156980 0.1684681 0.2340906 0.2289306 0.2937019 0.3854181    0
## ENET 0.06022713 0.1669587 0.2352686 0.2291473 0.2945983 0.3894504    0
## SVM  0.08315655 0.2606771 0.3688956 0.3586119 0.4755752 0.5686655    0
## RF1  0.10273136 0.2812357 0.3521249 0.3640580 0.4508770 0.6174405    0
## RF2  0.07474124 0.2672851 0.3399856 0.3383993 0.4312137 0.5510384    0
```



The errors of the second Random Forest model have more of a balance between bias and variance, the training error is slightly lower than the validation error, which itself is relatively low as well. This is the optimal zone we want to be in as the gap between the two errors is not too large and both values are low. We want a model that trains with great accuracy but that will also generalize well to new data. Based on the RMSE results, our best choice for the final hold-out evaluation is the second Random Forest model.

4.7.1 Preprocessing the test set

First, we will preprocess the test set using the same preprocessing steps we took to prepare the training set. The following code achieves this:

```
# Preprocessing of training set

# Feature Selection
vgTest<-vgTest%>%
  dplyr::select(-c(User_Score,Publisher,Developer,Rating,Genre,Platform_Type,Name,Year))
colnames(vgTest) #check column names
```

```
## [1] "Platform"     "Global_Sales" "Critic_Score" "Critic_Count" "User_Count"
## [6] "Age"
```

```
nrow(vgTest) #check number of observations
```

```
## [1] 3345
```

```r
# Check if any NAs in datasets
apply(vgTest,2,function(x){sum(is.na(x))})
```

```
##     Platform Global_Sales Critic_Score Critic_Count   User_Count          Age
##            0            0         1717         1717         1846           58
```

```r
# Remove NAs
vgTest<-vgTest[complete.cases(vgTest),]
nrow(vgTest)
```

```
## [1] 1367
```

```r
# Rename to test set
test_set<-vgTest

# Center and scale numeric features
test_set<-predict(preProcValues, test_set)

# Dummify categorical variables in test set
t2_dummy <- dummyVars(" ~ .", data = test_set)
test_set_dummy <- data.frame(predict(t2_dummy, newdata = test_set))
head(test_set_dummy)
```

	Platform.2600	Platform.3DO	Platform.3DS	Platform.DC	Platform.DS	Platform.GB	Platform.GBA	Pla
3	0	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	
27	0	0	0	0	1	0	0	
38	0	0	0	0	0	0	0	
39	0	0	0	0	0	0	0	
55	0	0	0	0	0	0	0	

4.7.2 Baseline Model

For point of comparison, we will build a baseline model that predicts global sales using just the mean, as follows:

```r
# Baseline Model - Guessing Global Sales based on the mean
set.seed(1234)

# Mean of observed Global Sales
mu<-mean(train_set$Global_Sales)
mu
```

```
## [1] 3.906182e-17
```

```r
# Predict all unknown ratings with overall mean

naive_rmse <- RMSE(test_set$Global_Sales, mu)
naive_rmse
```

```
## [1] 0.8664683
```

Any final model we implement should perform better than this.

Table 1: RMSE Results

Method	Training Error	Validation Error	RMSE
Linear Regression	0.9088346	0.8335534	NA
Elastic Net	0.9094815	0.8330187	NA
SVM	0.8618670	0.7910238	NA
RandomForest 1	0.4640654	0.7726072	NA
RandomForest 2	0.7668527	0.7847341	NA
Baseline	NA	NA	0.8664683

4.7.3 Testing the Final Model

We start the final validation by training the best model (Random Forest #2) on all available data, which means we won't be conducting any resampling or cross-validation during this phase so the trainControl argument will be removed.

```
# Final model - Random Forest with default parameters

set.seed(1234)

# Train the final model using all available data
fit_final <- train(Global_Sales~.,
                   train_dummy,
                   method = "rf",
                   tuneGrid = tuneGrid,   # best mtry
                   importance = TRUE,
                   nodesize = 14,
                   ntree = 600,   # best ntree
                   maxnodes = 24,
                   metric=metric)  # best maxnodes

fit_final
```

```
## Random Forest
##
## 5475 samples
##   35 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 5475, 5475, 5475, 5475, 5475, 5475, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   0.870699  0.2570821  0.2916007
##
## Tuning parameter 'mtry' was held constant at a value of 10
```

Now we make predictions on the test set (unseen data) using the final model.

Table 2: RMSE Results

Method	Training Error	Validation Error	RMSE
Linear Regression	0.9088346	0.8335534	NA
Elastic Net	0.9094815	0.8330187	NA
SVM	0.8618670	0.7910238	NA
RandomForest 1	0.4640654	0.7726072	NA
RandomForest 2	0.7668527	0.7847341	NA
Baseline	NA	NA	0.8664683
Final Model	NA	NA	0.7238861

```r
# Predicting on test set
test_final<- predict(fit_final, test_set_dummy)

# Check model performance
RMSE(test_final, test_set_dummy$Global_Sales)
```

```
## [1] 0.7238861
```

```r
# Check variable importance
varImp(fit_final)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                Overall
## Platform.DS   100.000
## Critic_Count   96.594
## Platform.PC    80.083
## Platform.PS2   71.358
## Platform.PS    69.642
## Platform.Wii   55.907
## Critic_Score   49.466
## Platform.PS3   47.559
## User_Count     47.538
## Age            43.324
## Platform.PSV   38.582
## Platform.PS4   31.748
## Platform.X360  31.741
## Platform.XOne  30.136
## Platform.DC    30.109
## Platform.GC    25.713
## Platform.GBA   24.310
## Platform.3DS   14.099
## Platform.XB     9.118
## Platform.PSP    8.478
```

We will now evaluate the results to check how the final algorithm performed compared to the other models:

RESULTS

The final model outperformed the validation error produced by Random Forest #2 during the training phase, minimizing the RMSE even further, which is a good sign that the model generalizes well to unseen data. This means the model is flexible enough to produce accurate predictions as new data gets introduced.

5 Conclusion

We started off our project by preparing our dataset and analyzing it for any insights that might be helpful in building our predictor model. From our analysis, we saw that some features had a stronger correlation with the target variable than others. During the feature selection stage, we finalized 5 features (User Count, Critic Count, Critic Score, Age, and Platform) to input into our machine learning models.

We started off by building a simple linear regression model. Although this was a good starting point, it produced quite a high validation error. We then used Elastic Net Regression, which implements L1 and L2 regularization by penalizing coefficients. This did not have a significant effect on the RMSE. We then trained an SVM Radial model which produced much better results than the previous two models. Finally, we trained two Random Forest models, one with default parameters and one with tuned parameters. The second model outperformed all the other models producing a low training error and a low validation error, with a small gap between the two numbers. This is the model we chose to make our final predictions.

Finally, we evaluated the performance of our model by training on the whole dataset and making predictions on the test set (unseen data). The model performed even better on the new data which confirmed that it was an optimal choice.

This project showed that the key features in determining global sales of video games are user count, critic count, critic score, age of the game, and the platform. This information could be used by gaming companies to make better decisions about how to develop, design, market and sell games globally.

5.1 Limitations

There were a few limitations when working with this dataset, such as the large proportion of missing values (over 50%) in key features that could have helped make more accurate predictions. There was also no data since 2016 so the information was slightly outdated and may have skipped more recent platforms. From a data collection perspective, the dataset did not cover mobile phone videogames, which is the largest contributor to global video games sales and also the fastest growing segment [39]. PC games were also not covered extensively in this dataset, another major contributor to global video game sales.

Our model did not take into account that some games are released across multiple platforms (Playstation and XBOX) while others are released exclusively to one platform (Nintendo). This could have an impact on predictions.

The Random Forest models, although robust and better at generalizing to new data, take a long time to run. This may be an issue when working with larger datasets.

5.2 Future Work

The goal of this report was to predict global sales for video games based on various features in the dataset. We did this using four different machine learning algorithms – Linear Regression, Elastic Net, Support Vector Machine, and Random Forest, although there are others such as k-Nearest Neighbors.

Some ideas for future work include:

- Updating the dataset for years 2016-2020 by either web scraping or data collection, and then running our final model again to see how it performs.

[39]https://www.reuters.com/article/esports-business-gaming-revenues-idUSFLM8jkJMl

- Feature Engineering #1 – creating a new feature for Console Type (home or portable), with the portable category including mobile phones, to check whether this would have an impact on predicting sales.
- Feature Engineering #2 – creating a new feature to separate exclusive games from multi-platform ones. Again this would be to check whether our model could make better predictions.
- Data Imputation – impute the missing values and run our models again to check whether they will perform even better.
- Ensemble Modeling – building and training an ensemble of all our models to check whether this performs better than our final model.