Video Game Analysis

Kashish Pandey

Overall Objectives

This project utilizes a video game dataset containing games from 1990 - 2016 from Kaggle. The first portion of this project jumps into EDA (exploratory data analysis). Essentially, I wanted to analyze the data before performing any models on it. I also wanted to fully clean up the model and plot the basic information out first to understand what was going on; I dropped missing values, rescaled the axis, fixed the scaling of variables, and mutated some columns in order to do so.

In terms of the second portion, I wanted to see what types of models could best predict global sales. The variables used for the model were Critic_Score,User_Score,Genre,Year_of_Release,Critic_Count, User_Count, Rating, publisher_top, developer_top,num_of_platform while predicting global sales. I used Linear Regression, Support Vectors Machines, Lasso, Ridge, and Random Forest models. I decided to use the metric of RMSE (root mean square error) to compare these models because it was a concise way to measure the actual values versus predicted(the outcome). I was able to see which models were overfitting and underfitting. Through hyperparameter tuning I was able to find optimal values to further improve the models!

Importing Libraries

- The script at the beginning insures that you have all the packages needed to run the following code! It is followed by importing the libraries once they are all downloaded.
- Optional: You can rerun this portion of code once everything is downloaded to get the message “0 packages had to be installed.”

```
my_packages <- c("tidyverse","testthat","ggplot2", "tree","caret","elasticnet",
                 "corrplot","kernlab")
not_installed <- my_packages[!(my_packages %in% installed.packages()[ , "Package"])]
if(length(not_installed)) install.packages(not_installed)
print(paste(length(not_installed), "packages had to be installed."))
```

```
## [1] "0 packages had to be installed."
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.0.2
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.3     ✓ purrr   0.3.4
## ✓ tibble  3.0.5     ✓ dplyr   1.0.3
## ✓ tidyr   1.1.2     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.1
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

```
## Warning: package 'tibble' was built under R version 4.0.2
```

```
## Warning: package 'tidyr' was built under R version 4.0.2
```

```
## Warning: package 'readr' was built under R version 4.0.2
```

```
## Warning: package 'purrr' was built under R version 4.0.2
```

```
## Warning: package 'dplyr' was built under R version 4.0.2
```

```
## Warning: package 'stringr' was built under R version 4.0.2
```

```
## Warning: package 'forcats' was built under R version 4.0.2
```

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

```
library(testthat)
```

```
## Warning: package 'testthat' was built under R version 4.0.2
```

```
## 
## Attaching package: 'testthat'
```

```
## The following object is masked from 'package:dplyr':
## 
##     matches
```

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

```
## The following object is masked from 'package:tidyr':
## 
##     matches
```

```
# For plotting
library(ggplot2)
# Random Forest Model
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.0.2
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
# Regression Model
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.2
```

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

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

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

```
# Lasso and Ridge Models
library(elasticnet)
```

```
## Warning: package 'elasticnet' was built under R version 4.0.2
```

```
## Loading required package: lars
```

```
## Warning: package 'lars' was built under R version 4.0.2
```

```
## Loaded lars 1.2
```

```
# Correlation Plot
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.0.2
```

```
## corrplot 0.84 loaded
```

```
# SVM Models (linear,poly,radial)
library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 4.0.2
```

```
##
## Attaching package: 'kernlab'
```

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

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

Exploratory Data Analysis

- Reading the file
- Na.strings is removing null/blank values within the dataset

```
vg_sales <- read.csv("dataset/Video_Games_Sales_as_at_22_Dec_2016.csv",
                        sep=",",na.strings=c(""," ","NA","N/A"))
```

- Viewing the first 5 lines of the csv file

```
head(vg_sales)
```

```
##                        Name Platform Year_of_Release        Genre Publisher
## 1               Wii Sports      Wii            2006       Sports  Nintendo
## 2        Super Mario Bros.      NES            1985     Platform  Nintendo
## 3           Mario Kart Wii      Wii            2008       Racing  Nintendo
## 4        Wii Sports Resort      Wii            2009       Sports  Nintendo
## 5 Pokemon Red/Pokemon Blue       GB            1996 Role-Playing  Nintendo
## 6                   Tetris       GB            1989       Puzzle  Nintendo
##   NA_Sales EU_Sales JP_Sales Other_Sales Global_Sales Critic_Score Critic_Count
## 1    41.36    28.96     3.77        8.45        82.53           76           51
## 2    29.08     3.58     6.81        0.77        40.24           NA           NA
## 3    15.68    12.76     3.79        3.29        35.52           82           73
## 4    15.61    10.93     3.28        2.95        32.77           80           73
## 5    11.27     8.89    10.22        1.00        31.37           NA           NA
## 6    23.20     2.26     4.22        0.58        30.26           NA           NA
##   User_Score User_Count Developer Rating
## 1        8.0        322  Nintendo      E
## 2         NA         NA      <NA>   <NA>
## 3        8.3        709  Nintendo      E
## 4        8.0        192  Nintendo      E
## 5         NA         NA      <NA>   <NA>
## 6         NA         NA      <NA>   <NA>
```

- Checking total number of null values within the dataset

```
colSums(is.na(vg_sales))
```

```
##            Name        Platform Year_of_Release           Genre       Publisher
##               2               0             269               2              54
##        NA_Sales        EU_Sales        JP_Sales     Other_Sales    Global_Sales
##               0               0               0               0               0
##    Critic_Score    Critic_Count      User_Score      User_Count       Developer
##            8582            8582            9129            9129            6623
##          Rating
##            6769
```

Dropping NULL and NA values from the dataset

- There seem to be many missing values within this dataset
- This is because it is the combination of 2 different datasets and many of the original observations do not match the data from the second dataset
- Here I am dropping all the missing values

```
vg_sales <- vg_sales[complete.cases(vg_sales), ]
colSums(is.na(vg_sales))
```

```
##            Name        Platform Year_of_Release           Genre       Publisher
##               0               0               0               0               0
##        NA_Sales        EU_Sales        JP_Sales     Other_Sales    Global_Sales
##               0               0               0               0               0
##    Critic_Score    Critic_Count      User_Score      User_Count       Developer
##               0               0               0               0               0
##          Rating
##               0
```

- Analyzing the internal structure of each feature
- Noticed that user_score and critic_score are different structures but we will fix that after examining the dataset for outliers

```
str(vg_sales)
```

```
## 'data.frame':    6825 obs. of  16 variables:
##  $ Name           : chr  "Wii Sports" "Mario Kart Wii" "Wii Sports Resort" "New Super
Mario Bros." ...
##  $ Platform       : chr  "Wii" "Wii" "Wii" "DS" ...
##  $ Year_of_Release: int  2006 2008 2009 2006 2006 2009 2005 2007 2010 2009 ...
##  $ Genre          : chr  "Sports" "Racing" "Sports" "Platform" ...
##  $ Publisher      : chr  "Nintendo" "Nintendo" "Nintendo" "Nintendo" ...
##  $ NA_Sales       : num  41.4 15.7 15.6 11.3 14 ...
##  $ EU_Sales       : num  28.96 12.76 10.93 9.14 9.18 ...
##  $ JP_Sales       : num  3.77 3.79 3.28 6.5 2.93 4.7 4.13 3.6 0.24 2.53 ...
##  $ Other_Sales    : num  8.45 3.29 2.95 2.88 2.84 2.24 1.9 2.15 1.69 1.77 ...
##  $ Global_Sales   : num  82.5 35.5 32.8 29.8 28.9 ...
##  $ Critic_Score   : int  76 82 80 89 58 87 91 80 61 80 ...
##  $ Critic_Count   : int  51 73 73 65 41 80 64 63 45 33 ...
##  $ User_Score     : num  8 8.3 8 8.5 6.6 8.4 8.6 7.7 6.3 7.4 ...
##  $ User_Count     : int  322 709 192 431 129 594 464 146 106 52 ...
##  $ Developer      : chr  "Nintendo" "Nintendo" "Nintendo" "Nintendo" ...
##  $ Rating         : chr  "E" "E" "E" "E" ...
```

- Examining outlier data for sales

```
summary(vg_sales$NA_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0600  0.1500  0.3945  0.3900 41.3600
```

```
summary(vg_sales$EU_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0200  0.0600  0.2361  0.2100 28.9600
```

```
summary(vg_sales$JP_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## 0.00000 0.00000 0.00000 0.06416 0.01000 6.50000
```

```
summary(vg_sales$Other_Sales)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
##  0.00000  0.01000  0.02000  0.08268  0.07000 10.57000
```

```
summary(vg_sales$Global_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.0100  0.1100  0.2900  0.7776  0.7500 82.5300
```

- Examining outlier data for score/count

```
summary(vg_sales$Critic_Score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   13.00   62.00   72.00   70.27   80.00   98.00
```

```
summary(vg_sales$Critic_Count)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    3.00   14.00   25.00   28.93   39.00  113.00
```

```
summary(vg_sales$User_Count)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##     4.0    11.0    27.0   174.7    89.0 10665.0
```

```
summary(vg_sales$User_Score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.500   6.500   7.500   7.186   8.200   9.600
```

- Upon analysis, critic_score seems to be an int and user_score is num. These are two different structures and if we want to compare the two we need to make them the same
- Here I am changing the user_score to int to keep it consistent with critic_score

```
vg_sales$User_Score <- as.integer(vg_sales$User_Score)
summary(vg_sales$User_Score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.000   6.000   7.000   6.737   8.000   9.000
```

- It seems that critic_score and user_score are also out of different scales
- We need to put critic_score and user_score on the same scale
- user_score is only out of 10 and critic_score is out of 100
- By multiplying user_score by 10, both critic_score and critic_score are out of 100 now

```
vg_sales$User_Score <- vg_sales$User_Score * 10
```

- Here we need to alter the rating variable because there are only a few occurrences of the ratings “AO”,“K-A”, and “RP” once within the dataset.
- “AO” refers to Adult Only games so we can place that into the Mature Rating
- “K-A” refers to Kids to Adults so we can place that into the Everyone Rating
- “RP” refers to Rating Pending so we can place that into the Everyone Rating
- I mutated the column by adding AO, K-A, and RP into their own respective categories (either Mature rating and Everyone rating)

```
vg_sales %>% count(Rating)
```

```
##   Rating    n
## 1     AO    1
## 2      E 2082
## 3   E10+  930
## 4    K-A    1
## 5      M 1433
## 6     RP    1
## 7      T 2377
```

```
vg_sales <- vg_sales %>% mutate(Rating = ifelse(Rating == "AO", "M", Rating))
vg_sales <- vg_sales %>% mutate(Rating = ifelse(Rating == "K-A", "E", Rating))
vg_sales <- vg_sales %>% mutate(Rating = ifelse(Rating == "RP", "E", Rating))
vg_sales %>% count(Rating)
```

```
##   Rating    n
## 1      E 2084
## 2   E10+  930
## 3      M 1434
## 4      T 2377
```

Data Visualization

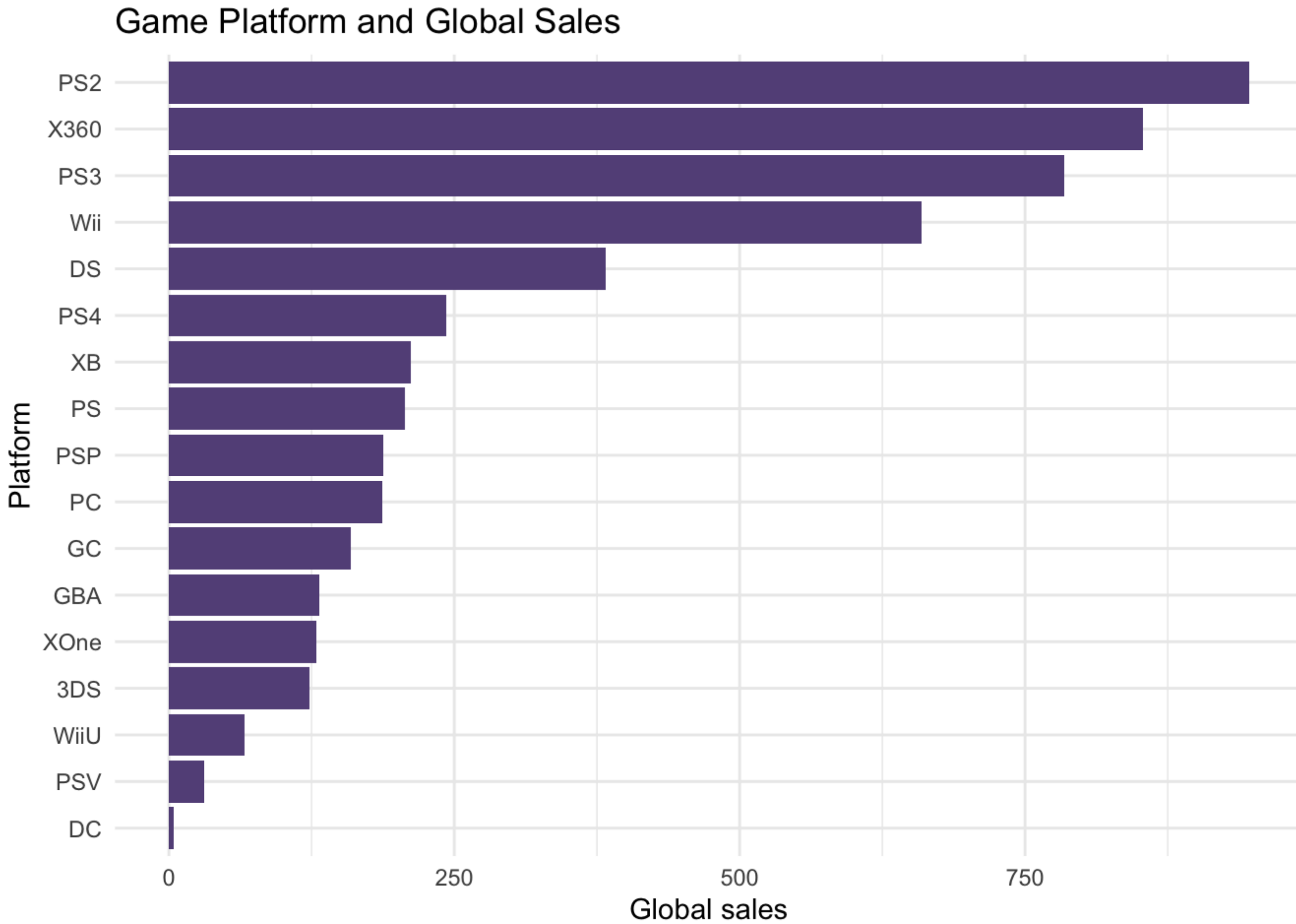
- Plotting game rating and global sales
- Teen games have the highest global sales!

```
rating_games_bar <- ggplot(vg_sales, aes(x = Rating,fill = Rating)) + geom_bar() +
      theme(text = element_text(size=10)) + xlab("Rating") + ylab("Global sales")+
  theme_minimal() + ggtitle("Game Rating and Global Sales")

rating_games_bar
```

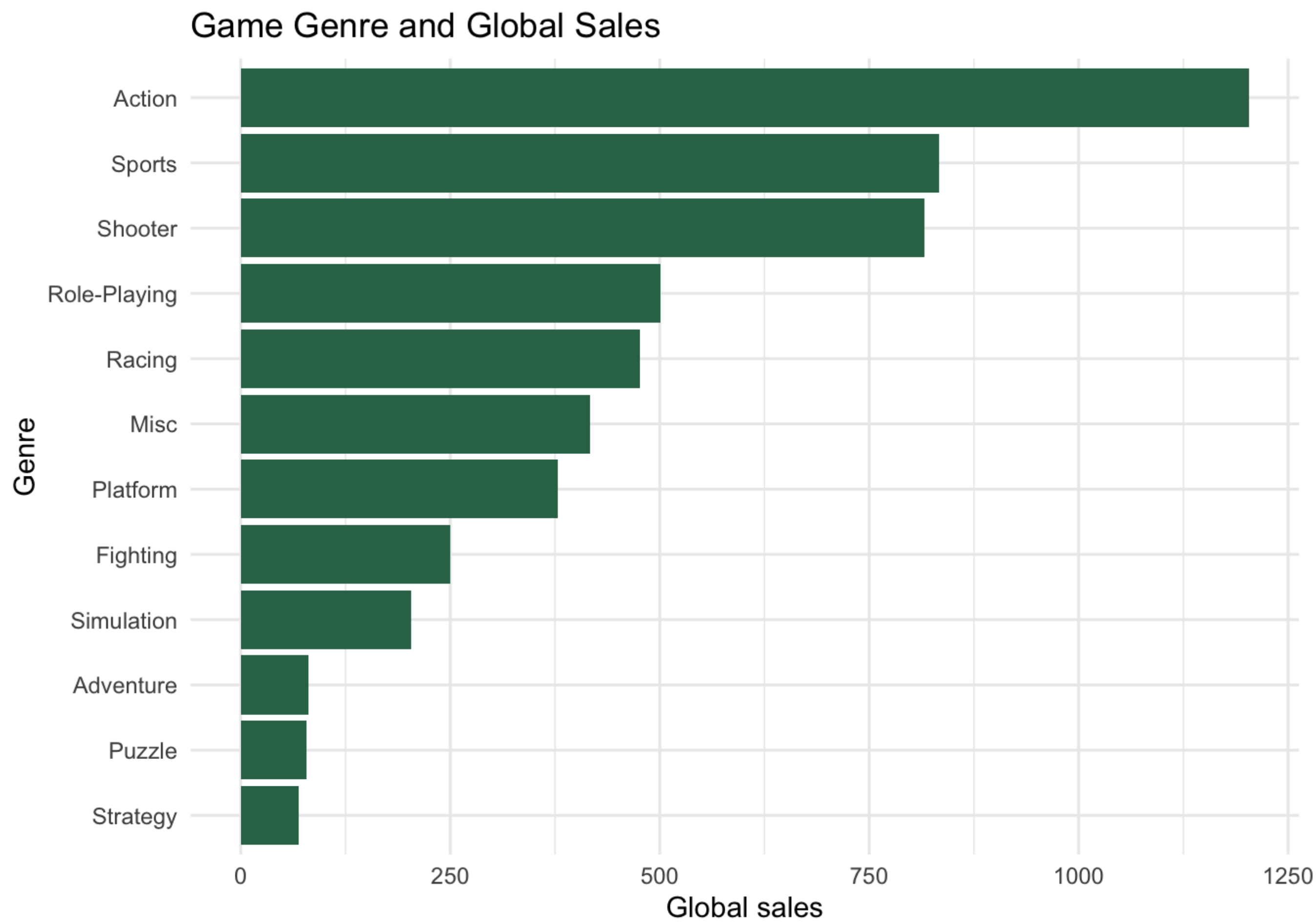
- The biggest global sales came from the platforms: Playstation 2 and Xbox360 followed by Playstation 3

```
vg_sales %>% group_by(Platform) %>%
  summarise(vg_sales = sum(Global_Sales)) %>% ggplot() +
  geom_bar(aes(reorder(Platform, vg_sales), vg_sales), stat = "identity",
           fill = "#645188") +
  xlab("Platform") + ylab("Global sales") +
  coord_flip() + theme_minimal() + ggtitle("Game Platform and Global Sales")
```



- Plotting genre and global sales
- The top genre are Action, Sport, and Shooter

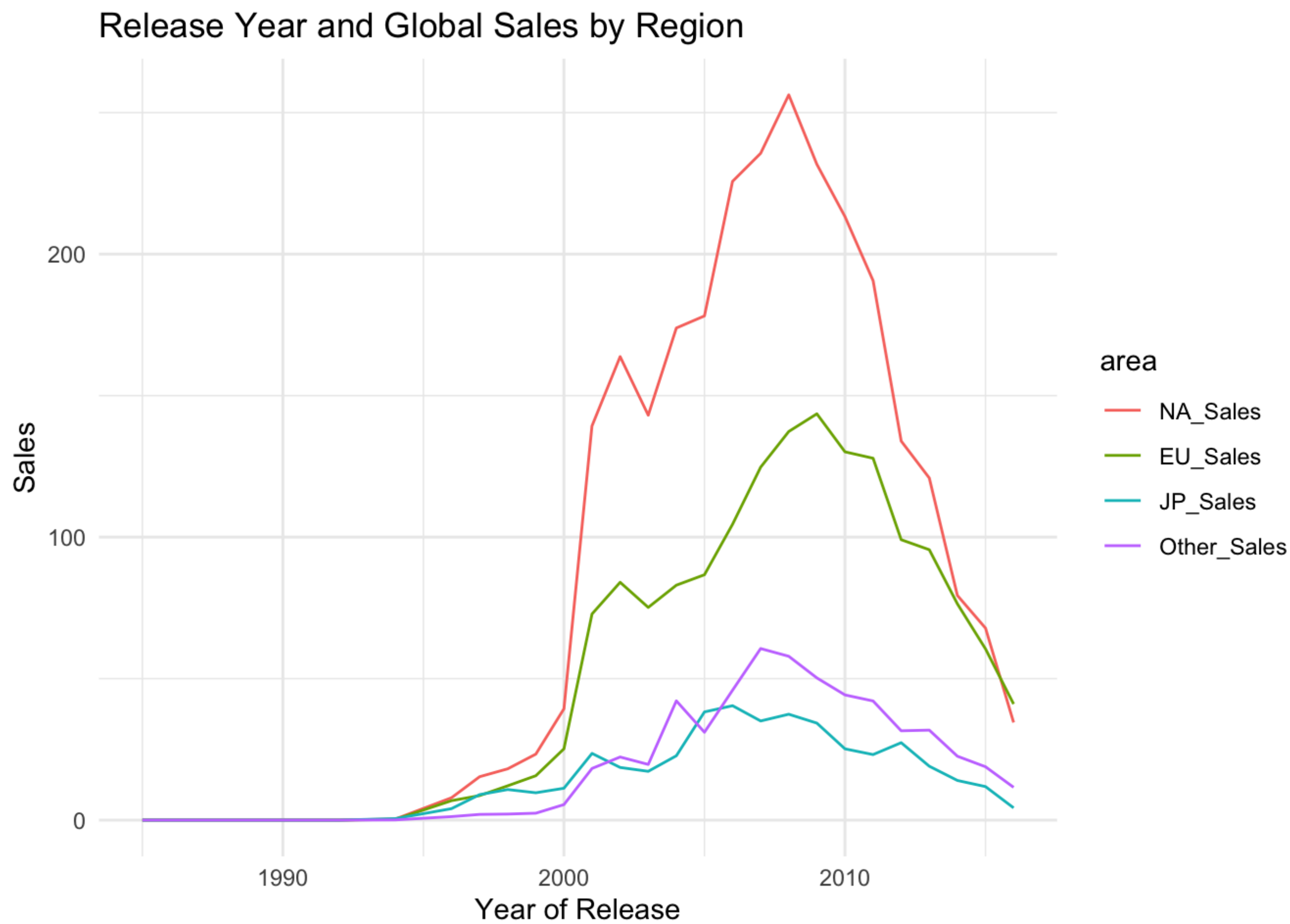
```
vg_sales %>% group_by(Genre) %>%
  summarise(vg_sales = sum(Global_Sales)) %>% ggplot() +
  geom_bar(aes(reorder(Genre, vg_sales), vg_sales), stat = "identity",
           fill = "#317256") +
  xlab("Genre") + ylab("Global sales") +
  coord_flip() + theme_minimal() +
  ggtitle("Game Genre and Global Sales")
```



- Plotting release year and global sales by North America, Europe, Japan, and Other
- Overall, North America had the highest sales from 1990-2016

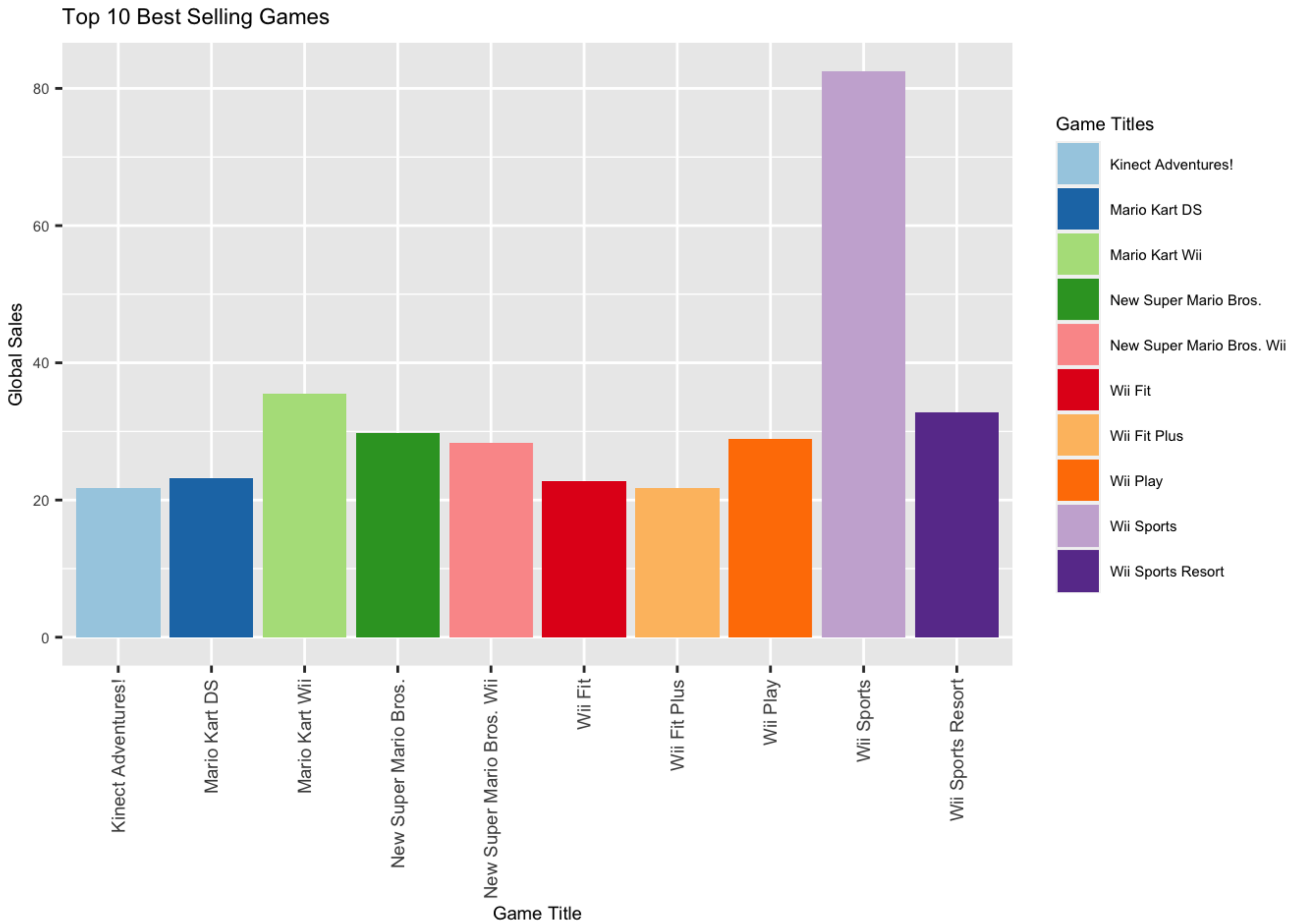
```
vg_sales %>% gather(area, vg_sales, NA_Sales:Other_Sales,
                  factor_key = TRUE) %>%
  group_by(area,Year_of_Release) %>%
  summarise(vg_sales = sum(vg_sales)) %>% ggplot() +
  xlab("Year of Release") + ylab("Sales") +
  geom_line(aes(Year_of_Release, vg_sales, group = area, color = area)) +
  theme_minimal() + theme(legend.text = element_text(size = 7),
                        legend.position = "bottom",
                        axis.text.x = element_text(angle = 90))+
  theme_minimal() + ggtitle("Release Year and Global Sales by Region")
```

```
## `summarise()` has grouped output by 'area'. You can override using the `.groups` argu
ment.
```



- Plotting the top 10 best selling games globally
- Wii sports is the #1 game sold globally

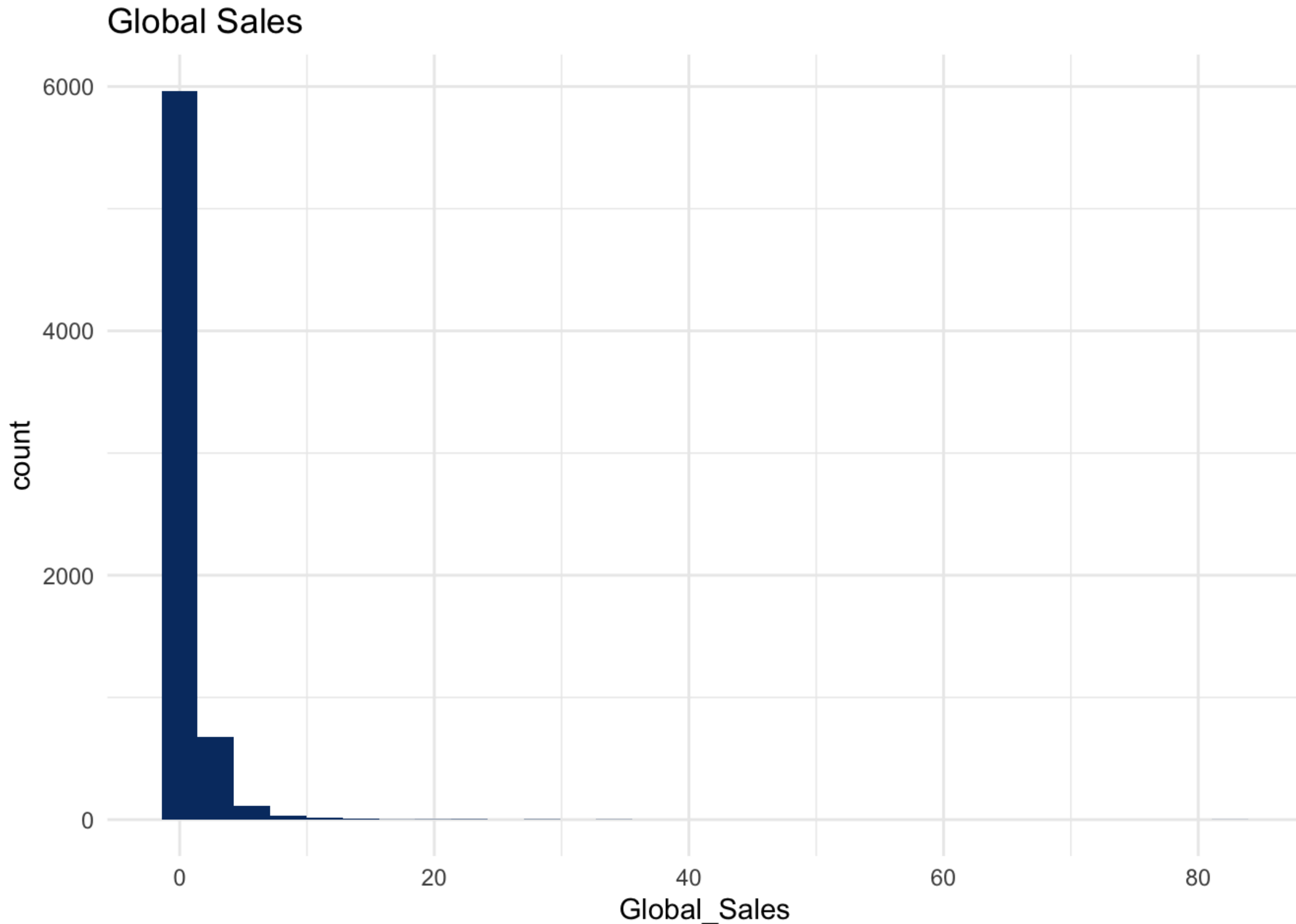
```
vg_sales %>% select(Name,Global_Sales) %>% arrange(desc(Global_Sales))%>% head(10)%>%
  ggplot(aes(x=Name,y=Global_Sales,fill= Name))+geom_bar(stat="identity")+
  labs(x="Game Title",y="Global Sales",
       title="Top 10 Best Selling Games")+
  theme(text = element_text(size=7),legend.position="right",
        axis.text.x=element_text(angle = 90,vjust = 0.5, hjust = 1,size=7))+
  scale_fill_brewer(name= "Game Titles", palette="Paired")
```



- Bar plot of global sales
- Overall, the plot is extremely skewed
- To fix this we need to change x axis to log axis because the distribution needs to be fixed

```
ggplot(vg_sales) + geom_histogram(aes(Global_Sales), fill = "#063970")+ ggtitle("Global
 Sales")+ theme_minimal()
```

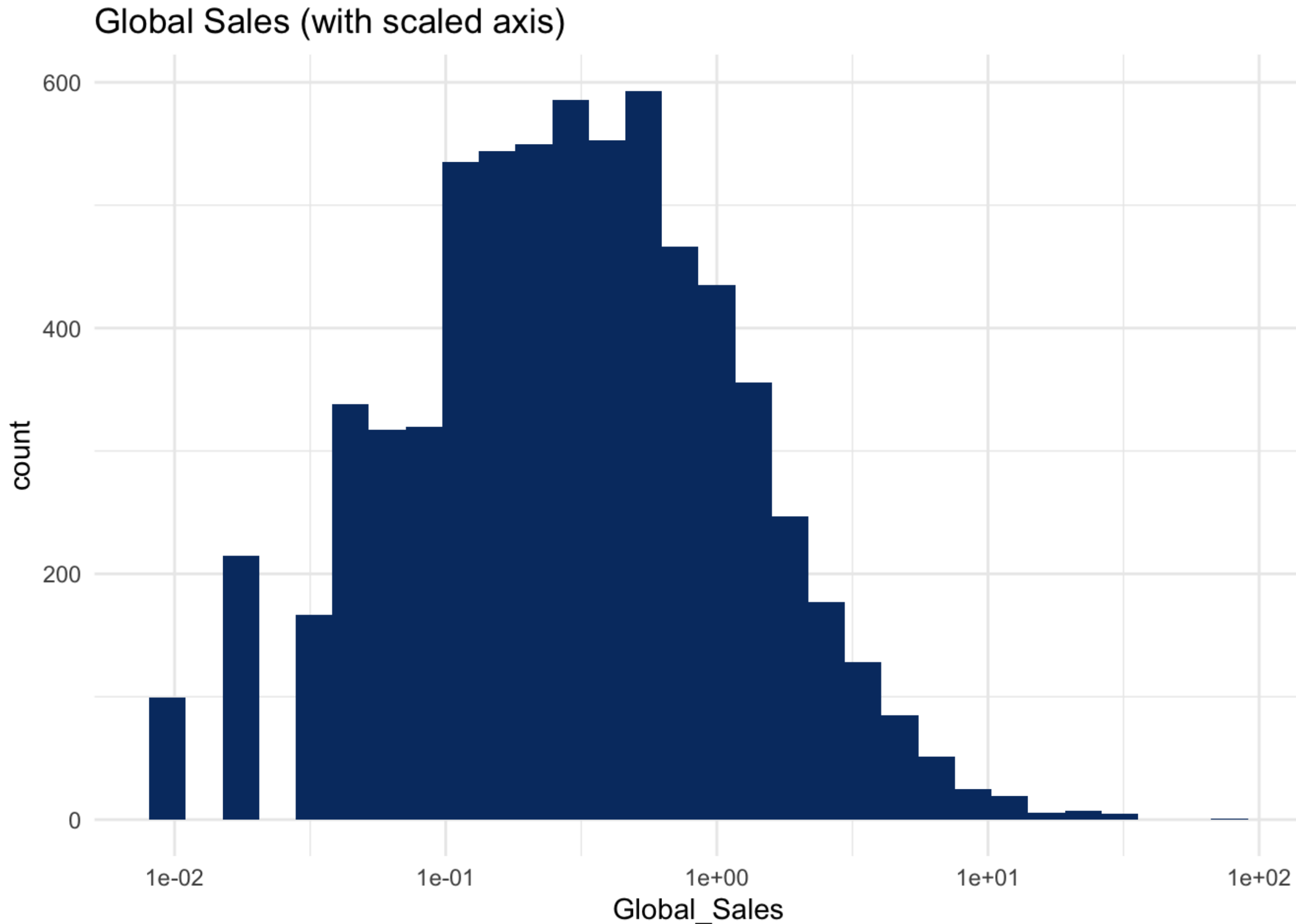
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



- By scaling the x axis to log axis it fixed the axis and provided a much better distribution (looks similar to a Gaussian distribution)

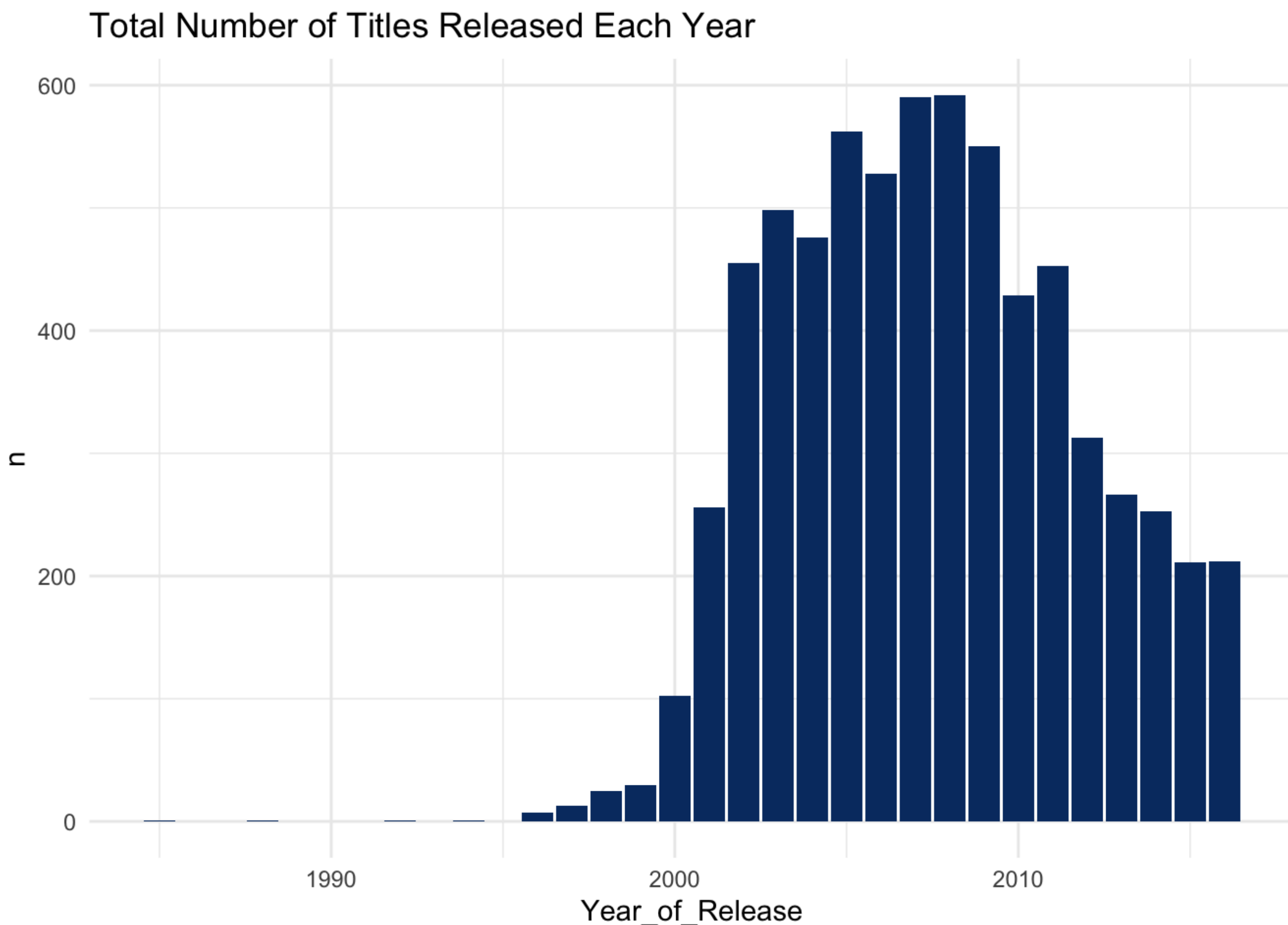
```
ggplot(vg_sales) + geom_histogram(aes(Global_Sales), fill = "#063970") +
  scale_x_log10() + ggtitle("Global Sales (with scaled axis)")+ theme_minimal()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



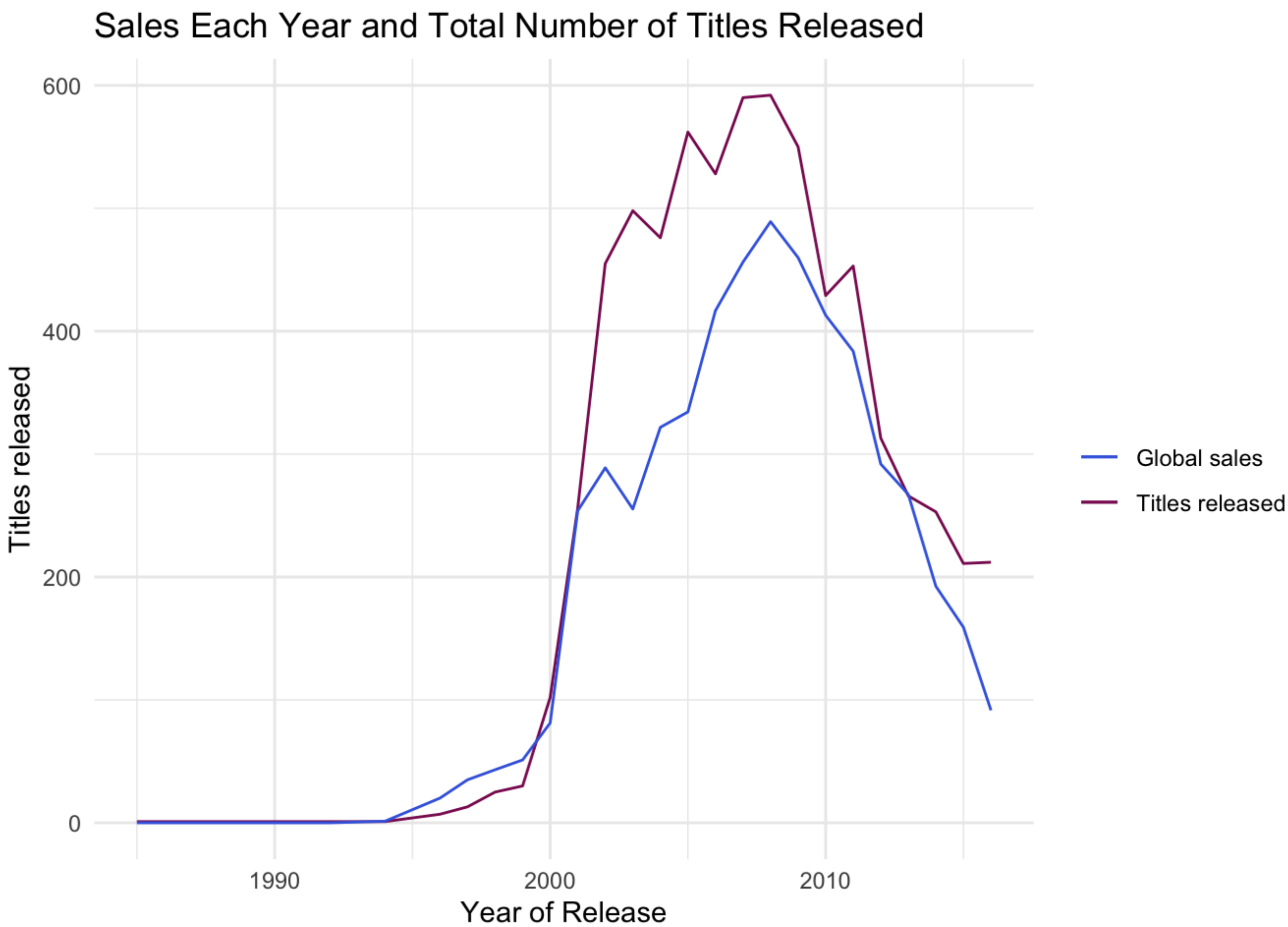
- Barplot of number of titles released each year
- There seems to be a peak within the data from 2005-2009 which just means there were a substantial amount of video hame titles released between those years

```
vg_sales %>% group_by(Year_of_Release) %>%
  count() %>% ggplot() +
  geom_bar(aes(Year_of_Release, n), stat = "identity",
           fill = "#063970") + theme(axis.text.x = element_text(angle = 90))+
  ggtitle("Total Number of Titles Released Each Year")+
  theme_minimal()
```



- Line graph of sales each year and total number of releases
- There is more revenue when more titles are released

```
color <- c("Titles released" = "maroon4", "Global sales" = "royalblue")
vg_sales %>% group_by(Year_of_Release) %>%
  summarise(vg_sales = sum(Global_Sales), count = n()) %>%
  ggplot() + xlab("Year of Release") + ylab("Titles released") +
  geom_line(aes(Year_of_Release, count, group = 1, color = "Titles released")) +
  geom_line(aes(Year_of_Release, vg_sales, group = 1, color = "Global sales")) +
  theme(axis.text.x = element_text(angle = 90), legend.position = "bottom") +
  scale_color_manual(name="",values = color) + theme_minimal() + ggtitle("Sales Each Yea
r and Total Number of Titles Released")
```



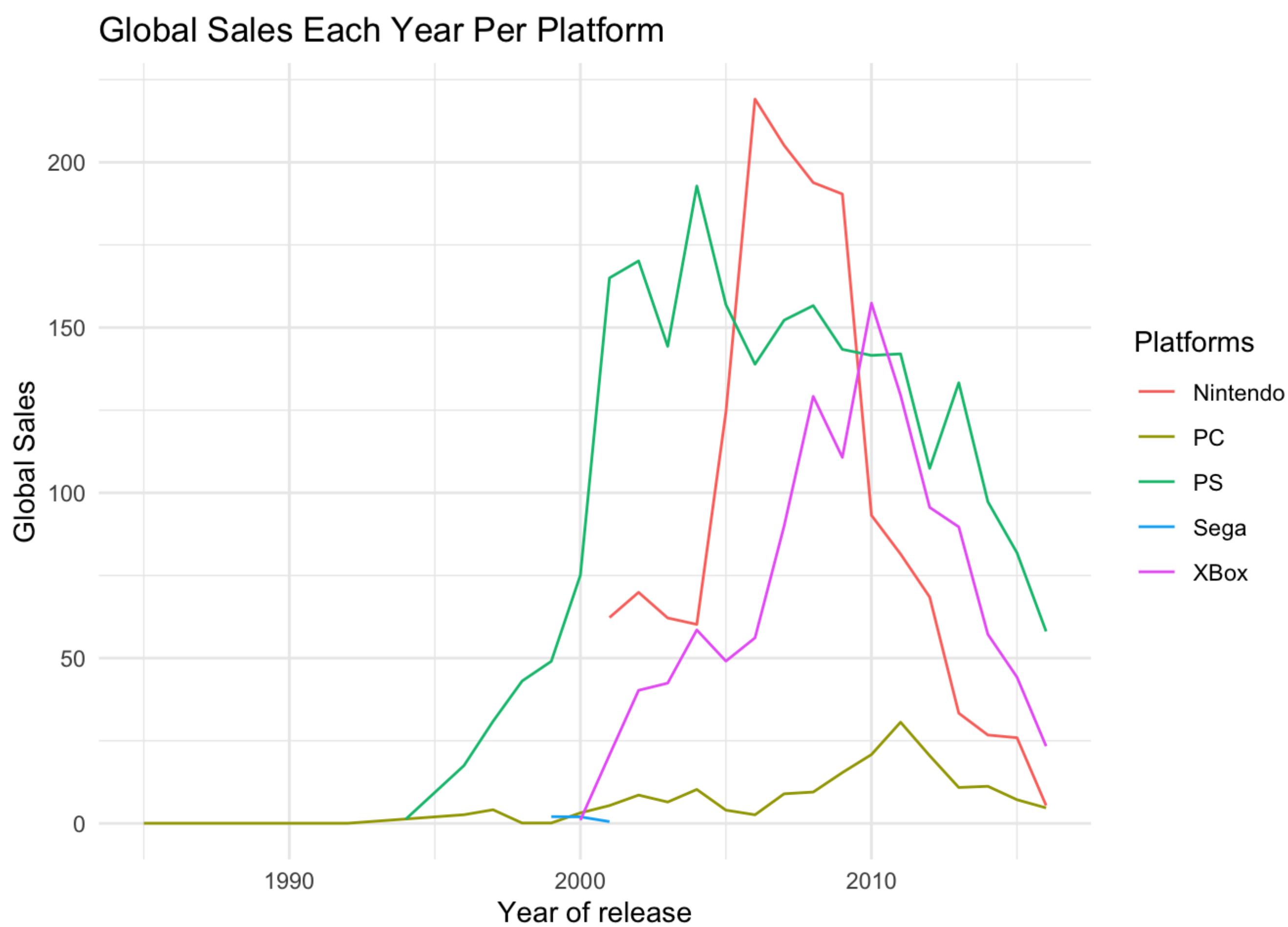
- To simplify the following graphs and to make the models easier to run later I am combining platforms by their respective company

```
vg_sales <- vg_sales %>% mutate(platform2 = case_when(
  Platform %in% c("Wii", "DS", "3DS", "WiiU", "GC", "GBA") ~ "Nintendo",
  Platform %in% c("X360", "XB", "XOne") ~ "XBox",
  Platform %in% c("PS3", "PS4", "PS2", "PS", "PSP", "PSV") ~ "PS",
  Platform == "PC" ~ "PC",
  Platform == "DC" ~ "Sega"
))
```

- Line graph of global sales each year for each platform
- Nintendo and Playstation both peaked near one another

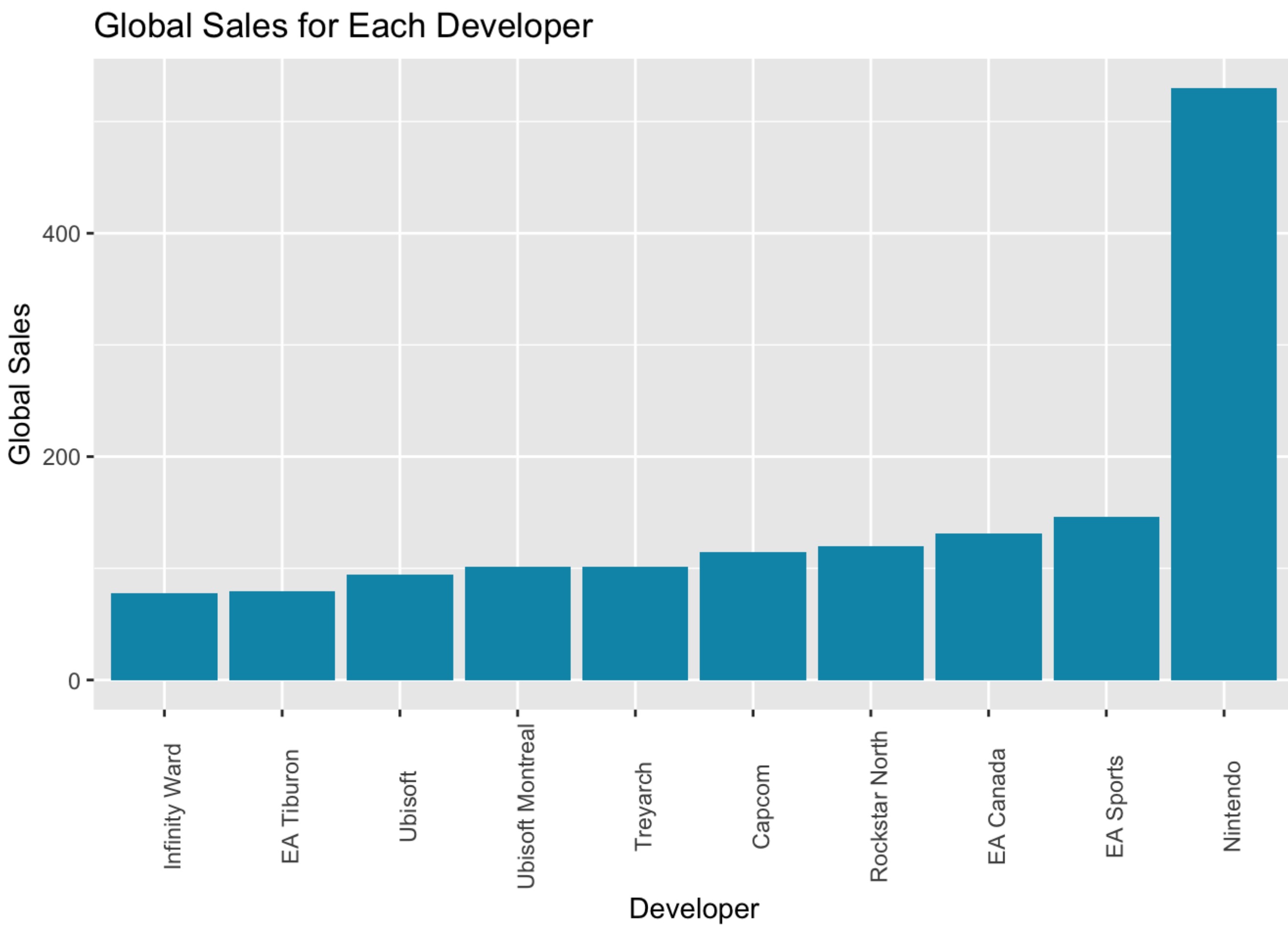
```
vg_sales %>% group_by(platform2, Year_of_Release) %>%
  summarise(vg_sales = sum(Global_Sales)) %>%
  ggplot() + xlab("Year of release") + ylab("Global Sales") +
  geom_line(aes(Year_of_Release, vg_sales, group = platform2, color = platform2)) +
  theme(legend.text = element_text(size = 7),
        axis.text.x = element_text(angle = 90, hjust = 1,
                                   vjust = 0.5, size = 6))+
  theme_minimal() + labs(color='Platforms') + ggtitle("Global Sales Each Year Per Platfo
rm")
```

```
## `summarise()` has grouped output by 'platform2'. You can override using the `.groups`
argument.
```



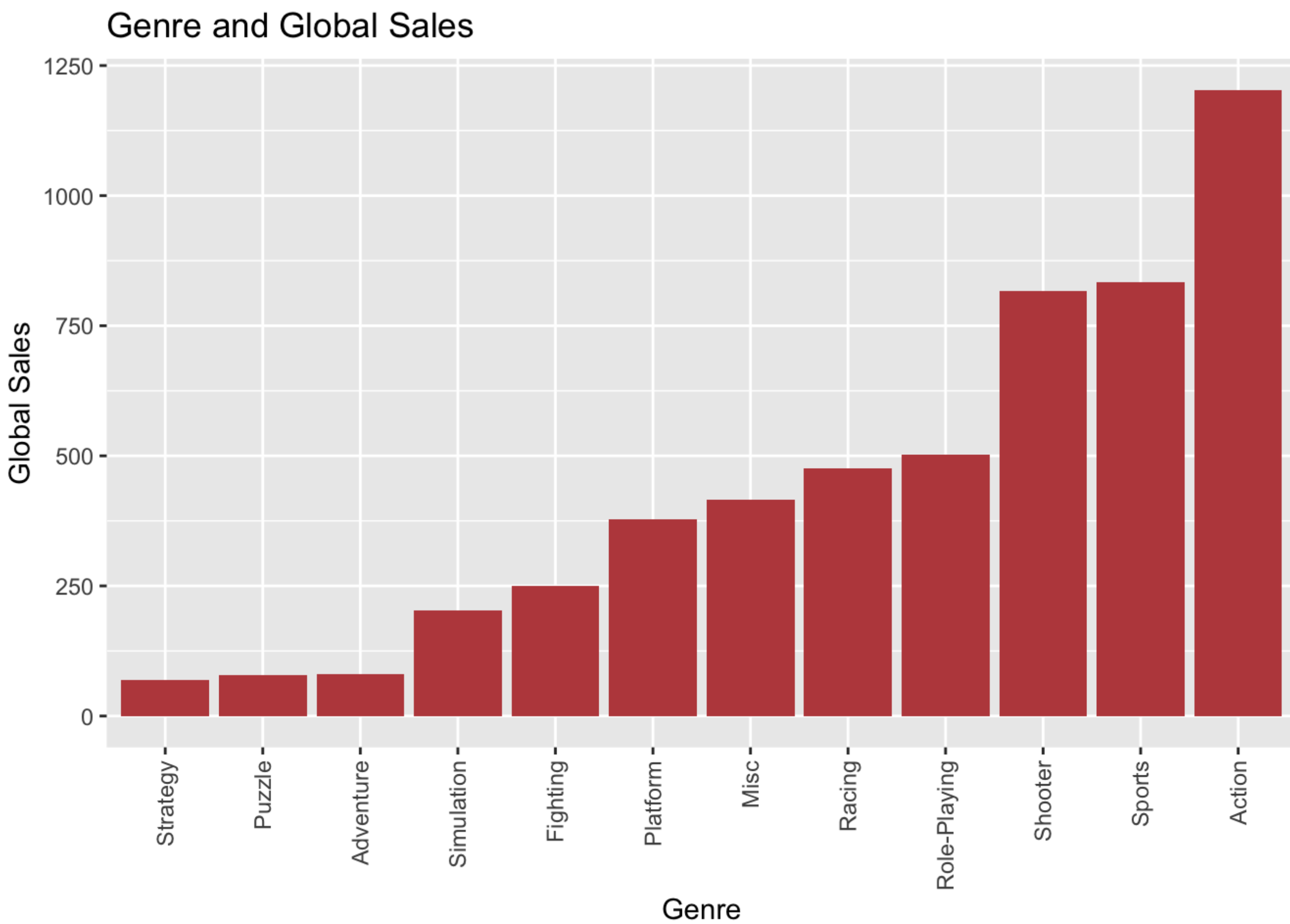
- Bar plot of sales for each developer and global sales
- Nintendo has the highest global sales across the board

```
vg_sales %>% group_by(Developer) %>%
  summarise(vg_sales = sum(Global_Sales)) %>%
  arrange(desc(vg_sales)) %>% slice(1:10) %>%
  ggplot() + xlab("Developer") + ylab("Global Sales")+
  geom_bar(aes(reorder(Developer, vg_sales), vg_sales),
           stat = "identity", fill = "#0095B6") +
  theme(axis.text.x = element_text(angle = 90)) +
  ggtitle("Global Sales for Each Developer")
```



- Bar plot of sales for each gaming genre and global sales

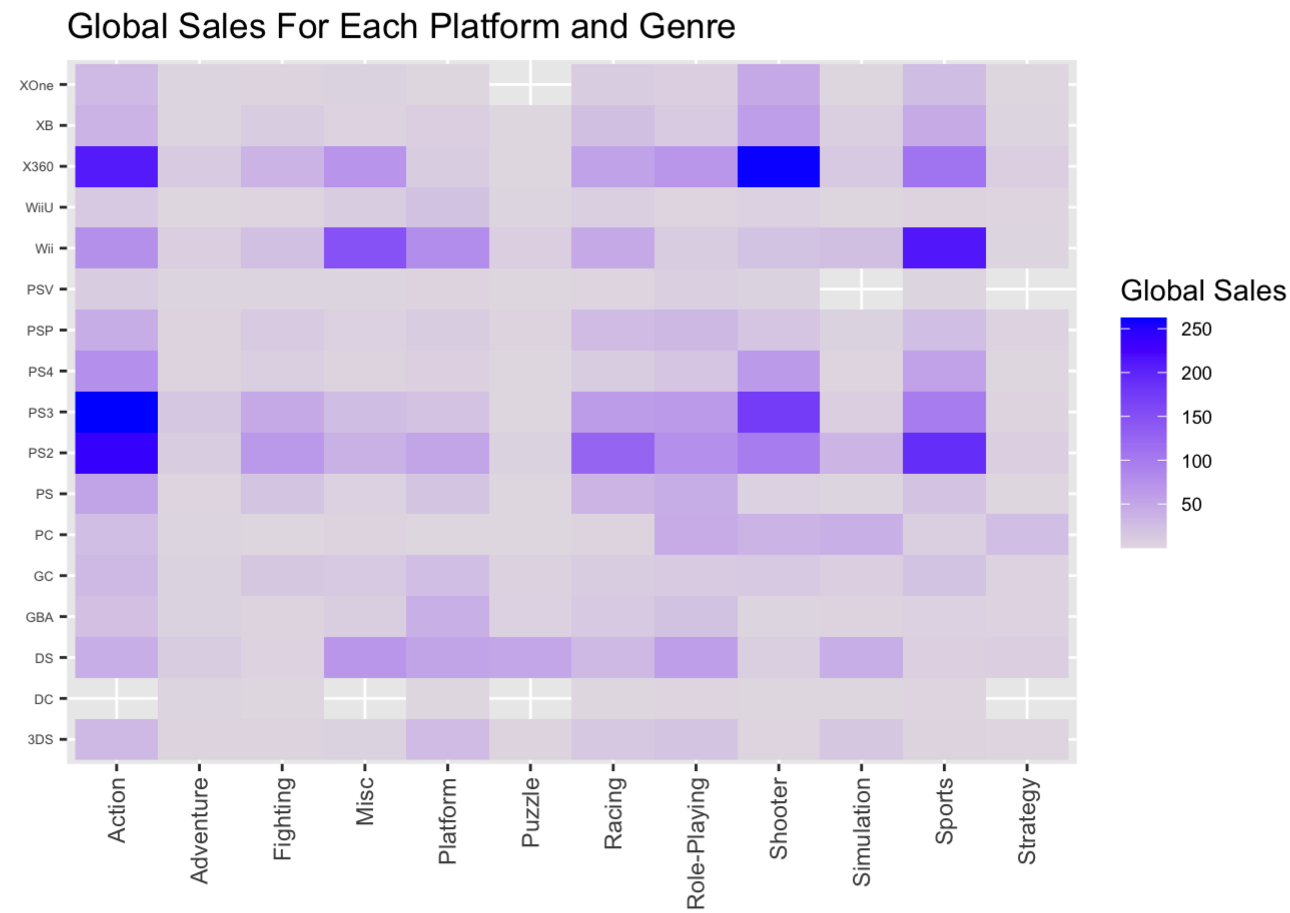
```
vg_sales %>% group_by(Genre) %>%
  summarise(vg_sales = sum(Global_Sales)) %>%
  ggplot() +
  geom_bar(aes(reorder(Genre, vg_sales), vg_sales), stat = "identity",
           fill = "#BC4B4B") +
  ylab("Global Sales") + xlab("Genre") +
  theme(axis.text.x = element_text(angle = 90,
                                   hjust = 1, vjust = 0.5)) +
  ggtitle("Genre and Global Sales")
```



- Correlation matrix of global sales for each platform and genre
- The top two global sales comes from Xbox 360 Shooter games and Playstation 3 Action games

```
vg_sales %>% group_by(Platform, Genre) %>%
  summarise(vg_sales = sum(Global_Sales)) %>%
  ggplot() + geom_raster(aes(Genre, Platform, fill = vg_sales)) +
  ylab("") + xlab("") +
  scale_fill_gradient(low = "#e4dee5", high = "blue") +
  theme(axis.text.x = element_text(angle = 90,
                                   vjust = 0.5, hjust = 1),
        axis.text.y = element_text(size = 5),
        legend.text = element_text(size = 7)) + labs(fill = "Global Sales")+
  ggtitle("Global Sales For Each Platform and Genre")
```

```
## `summarise()` has grouped output by 'Platform'. You can override using the `.groups`
argument.
```



Models for results

- Overall, the sales vary depending on the platform, release year, and developer
- The top developers had the highest sales
- Publishers is categorical variable but it contains has many values
- To combat this, we are selecting for only the top publishers

```
publishers_top <- (vg_sales %>% group_by(Publisher) %>%
                     summarise(vg_sales = sum(Global_Sales)) %>% arrange(desc(vg_sales))
%>%
                     top_n(10) %>% distinct(Publisher))$Publisher
```

```
## Selecting by vg_sales
```

- Developers is categorical variable but it contains has many values
- To combat this, we are selecting for only the top developers

```
developers_top <- (vg_sales %>% group_by(Developer) %>%
                     summarise(vg_sales = sum(Global_Sales)) %>% arrange(desc(vg_sales))
%>%
                     top_n(10) %>% distinct(Developer))$Developer
```

```
## Selecting by vg_sales
```

- Creating new variable for whether a game is created by a top developer/publisher
- Making it binary(0,1)

```
vg_sales <- vg_sales %>%
  mutate(publisher_top = ifelse(Publisher %in% publishers_top, TRUE, FALSE),
         developer_top = ifelse(Developer %in% developers_top, TRUE, FALSE))
```

- Checking whether games are exclusively launched on a specific platform

```
vg_sales <- vg_sales %>% group_by(Name) %>% mutate(num_of_platforms = n()) %>% ungroup(N
ame)
```

- Setting seed

```
set.seed(2000)
```

- Training and testing data sets
- Here I am setting the percentage of data that goes to training as 80% training and this would keep 20% for testing
- I tried 90-10, 80-20, and 70-30 for splitting of train/test but 80% seemed to improve RMSE the best

```
test_index <- createDataPartition(vg_sales$Global_Sales, p = 0.8, list = FALSE)
train_set <- vg_sales[-test_index, ]
test_set <- vg_sales[test_index, ]
```

- Including categorical data within the data

```
totalData <- rbind(train_set, test_set)
for (f in 1:length(names(totalData))) {
  levels(train_set[, f]) <- levels(totalData[, f])
}
```

Creating RMSE function

- RMSE refers to the Root Mean Square Error
- The Root Mean Square Error is the standard deviation of our predicted errors

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

Linear Regression Model

- This is my baseline model to generally understand the results of the model
- Baseline models are an important way to interpret the model with less complexity

```
model_lm <- train(log(Global_Sales) ~ Critic_Score +
                     User_Score + Genre +
                     Year_of_Release + Critic_Count +
                     User_Count + Rating +
                     publisher_top + developer_top +
                     num_of_platforms, method = "lm", data = train_set)

# predicted values and RMSE
test_set$predicted_lm <- predict(model_lm, test_set)
rmse_results <- data.frame(Method = "Linear Regression",
                           RMSE = RMSE(log(test_set$Global_Sales), test_set$predicted_l
m))
```

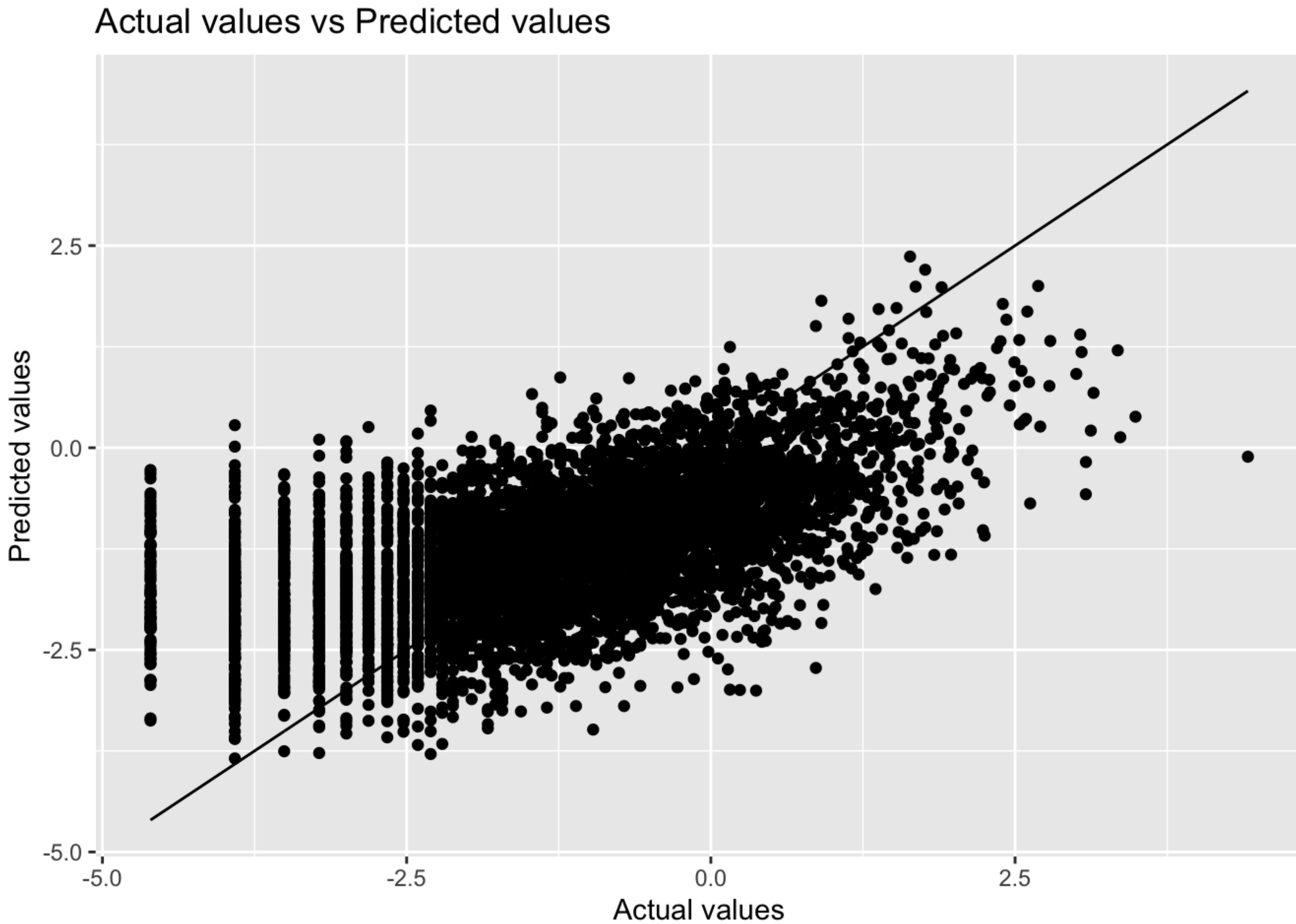
- Summary of linear regression model
- r^2: 0.3358
- This is not the best r^2 value, we generally want r^2 to be as close to 1 as possible

```
summary(model_lm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.4847 -0.6948  0.0720  0.7578  3.5810
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          5.358e+01  1.661e+01   3.225 0.001289 **
## Critic_Score         2.017e-02  3.178e-03   6.347 2.99e-10 ***
## User_Score          -1.796e-03  2.733e-03  -0.657 0.511308
## GenreAdventure      -7.300e-01  1.914e-01  -3.814 0.000143 ***
## GenreFighting        2.208e-01  1.587e-01   1.392 0.164194
## GenreMisc            3.281e-01  1.500e-01   2.187 0.028906 *
## GenrePlatform       -7.467e-02  1.542e-01  -0.484 0.628391
## GenrePuzzle         -5.277e-01  2.428e-01  -2.173 0.029952 *
## GenreRacing         -3.903e-01  1.331e-01  -2.931 0.003432 **
## `GenreRole-Playing` -4.425e-02  1.218e-01  -0.363 0.716339
## GenreShooter        -2.892e-01  1.164e-01  -2.485 0.013072 *
## GenreSimulation      3.819e-03  1.639e-01   0.023 0.981420
## GenreSports         -5.133e-01  1.286e-01  -3.990 6.97e-05 ***
## GenreStrategy       -1.184e+00  1.805e-01  -6.559 7.71e-11 ***
## Year_of_Release     -2.831e-02  8.267e-03  -3.425 0.000634 ***
## Critic_Count         2.379e-02  2.094e-03  11.360  < 2e-16 ***
## User_Count           1.851e-04  6.763e-05   2.737 0.006277 **
## `RatingE10+`        -3.256e-01  1.092e-01  -2.981 0.002923 **
## RatingM             -5.955e-01  1.239e-01  -4.808 1.70e-06 ***
## RatingT             -5.765e-01  9.821e-02  -5.870 5.48e-09 ***
## publisher_topTRUE    4.909e-01  7.065e-02   6.948 5.76e-12 ***
## developer_topTRUE    3.447e-01  1.064e-01   3.240 0.001226 **
## num_of_platforms     1.237e-01  2.537e-02   4.874 1.22e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.155 on 1340 degrees of freedom
## Multiple R-squared:  0.3383, Adjusted R-squared:  0.3275
## F-statistic: 31.14 on 22 and 1340 DF,  p-value: < 2.2e-16
```

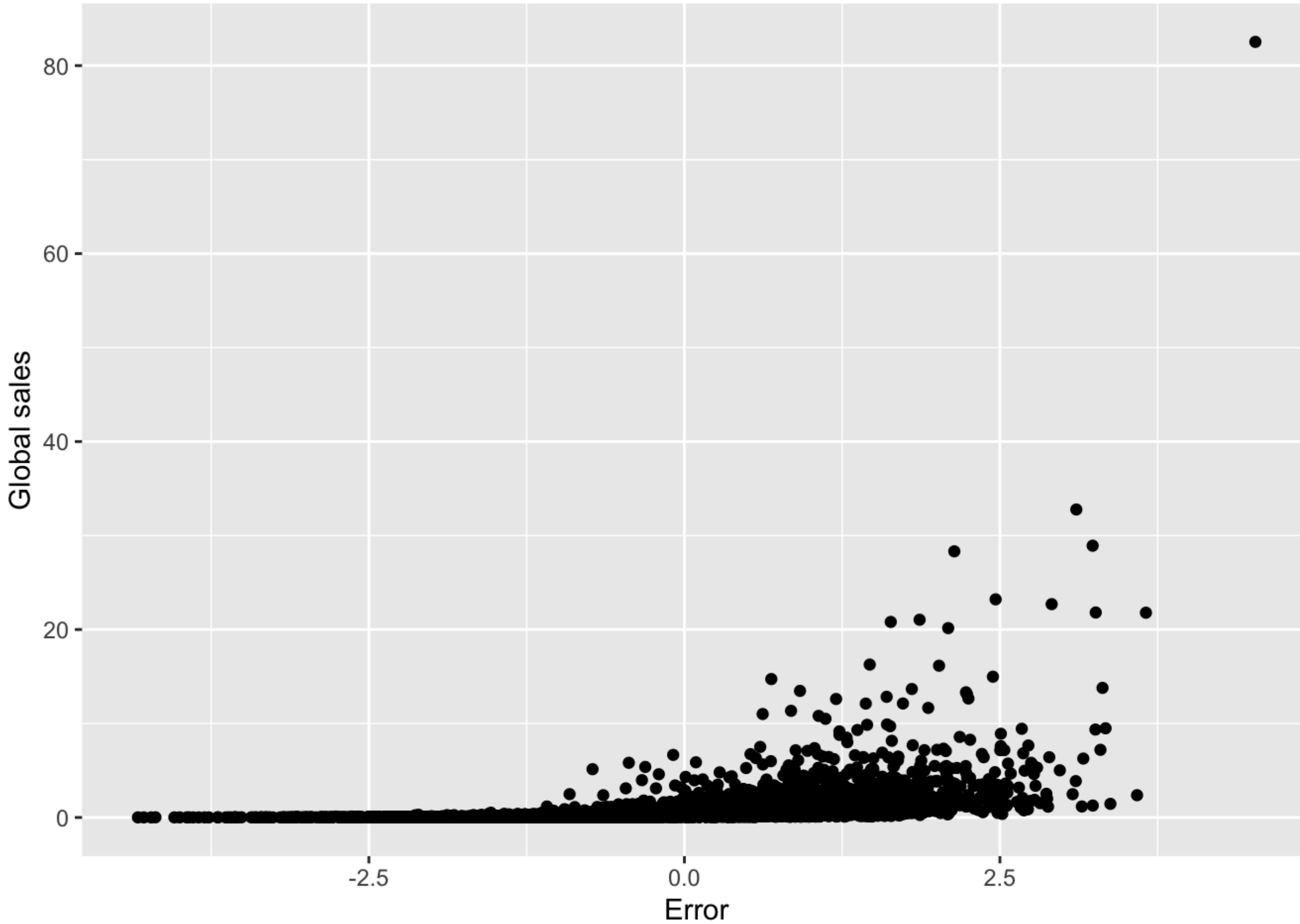
- Actual vs Predicted plot

```
ggplot(test_set) +
  geom_point(aes(log(Global_Sales), predicted_lm)) +
  geom_line(aes(log(Global_Sales), log(Global_Sales))) +
  xlab("Actual values") + ylab("Predicted values") +
  ggtitle("Actual values vs Predicted values")
```



- Residual plot (error vs predicted)
- Errors are largest for larger values of global sales this means that heteroskedacity present

```
ggplot(test_set) + geom_point(aes(log(Global_Sales) - predicted_lm, Global_Sales)) +
  xlab("Error") + ylab("Global sales")
```



SVM Linear Model

```
model_svm_linear <- train(log(Global_Sales) ~ Critic_Score +
                     User_Score + Genre +
                     Year_of_Release +  Critic_Count +
                     User_Count + Rating +
                     publisher_top + developer_top +
                     num_of_platforms, method = "svmLinear",
                   data = train_set)

# predicted value and RMSE
test_set$predicted_svm_linear <- predict(model_svm_linear, test_set)
rmse_results <- rmse_results %>%
  add_row(Method = "SVM Linear",
          RMSE = RMSE(log(test_set$Global_Sales),
                      test_set$predicted_svm_linear))
```

- Summary of SVM linear model

```
summary(model_svm_linear)
```

```
## Length  Class   Mode
##      1   ksvm     S4
```

SVM Poly Model

- This will take several minutes to run because it is more mathematically complex (polynomial function)

```
model_svm_poly <- train(log(Global_Sales) ~ Critic_Score +
                     User_Score + Genre +
                     Year_of_Release + Critic_Count +
                     User_Count + Rating +
                     publisher_top + developer_top +
                     num_of_platforms, method = "svmPoly",
                   data = train_set)

# predicted value and RMSE
test_set$predicted_svm_poly <- predict(model_svm_poly, test_set)
rmse_results <- rmse_results %>%
  add_row(Method = "SVM Polynomial",
          RMSE = RMSE(log(test_set$Global_Sales),
                      test_set$predicted_svm_poly))
```

- SVM Poly Model Summary

```
summary(model_svm_poly)
```

```
## Length  Class   Mode
##      1   ksvm     S4
```

SVM Radial Model

```
model_svm_rad <- train(log(Global_Sales) ~ Critic_Score +
                          User_Score + Genre +
                          Year_of_Release +  Critic_Count +
                          User_Count + Rating +
                          publisher_top + developer_top +
                          num_of_platforms, method = "svmRadial",
                        data = train_set)

# predicted value and RMSE
test_set$predicted_svm_rad<- predict(model_svm_rad, test_set)
rmse_results <- rmse_results %>%
  add_row(Method = "SVM Radial",
          RMSE = RMSE(log(test_set$Global_Sales),
                      test_set$predicted_svm_rad))
```

- SVM Radial Summary

```
summary(model_svm_rad)
```

```
## Length  Class   Mode
##      1   ksvm     S4
```

L1 - Lasso Model

- Lasso regression is essentially regularizated linear regression
- Compared to ridge regression, instead of penalizing high values, the lasso model sets these values equal to zero instead
- There is a chance to end up with fewer features because of the method lasso uses (it is essentially keeping the more important features) and this is where lasso can have an upperhand over ridge

```
model_l1 <- train(log(Global_Sales) ~ Critic_Score +
                    User_Score + Genre +
                    Year_of_Release +  Critic_Count +
                    User_Count + Rating +
                    publisher_top + developer_top +
                    num_of_platforms, method = "lasso", data = train_set)

# predicted values and RMSE
test_set$predicted_l1 <- predict(model_l1, test_set)
rmse_results <- rmse_results %>% add_row(Method = "L1 Lasso",
                                          RMSE = RMSE(log(test_set$Global_Sales),
                                                      test_set$predicted_l1))
```

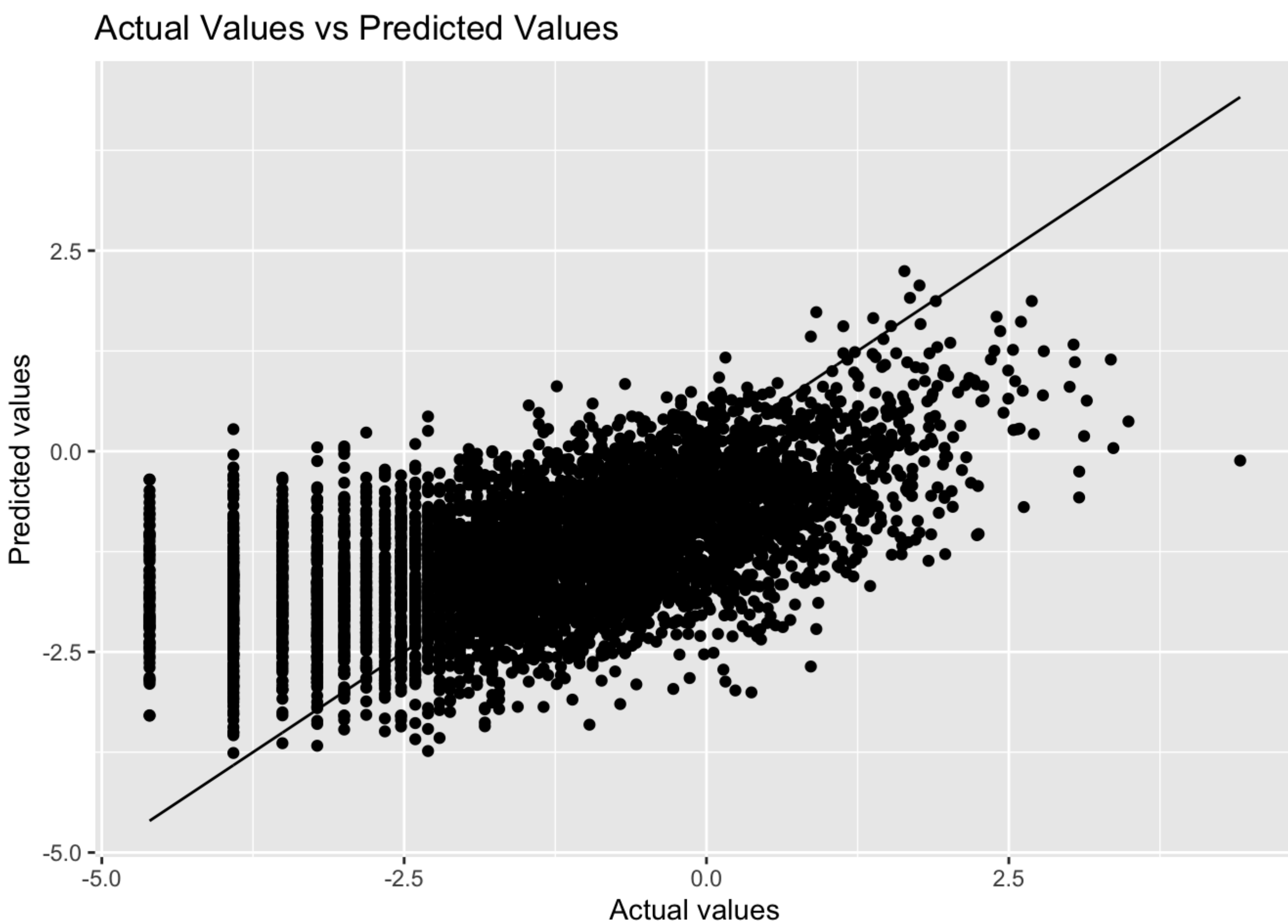
- Summary of Lasso Model

```
summary(model_l1)
```

```
##             Length Class      Mode
## call          4    -none-     call
## actions      25    -none-     list
## allset       22    -none-     numeric
## beta.pure   550    -none-     numeric
## vn           22    -none-     character
## mu            1    -none-     numeric
## normx        22    -none-     numeric
## meanx        22    -none-     numeric
## lambda        1    -none-     numeric
## L1norm       25    -none-     numeric
## penalty      25    -none-     numeric
## df           25    -none-     numeric
## Cp           25    -none-     numeric
## sigma2        1    -none-     numeric
## xNames       22    -none-     character
## problemType   1    -none-     character
## tuneValue     1    data.frame list
## obsLevels     1    -none-     logical
## param         0    -none-     list
```

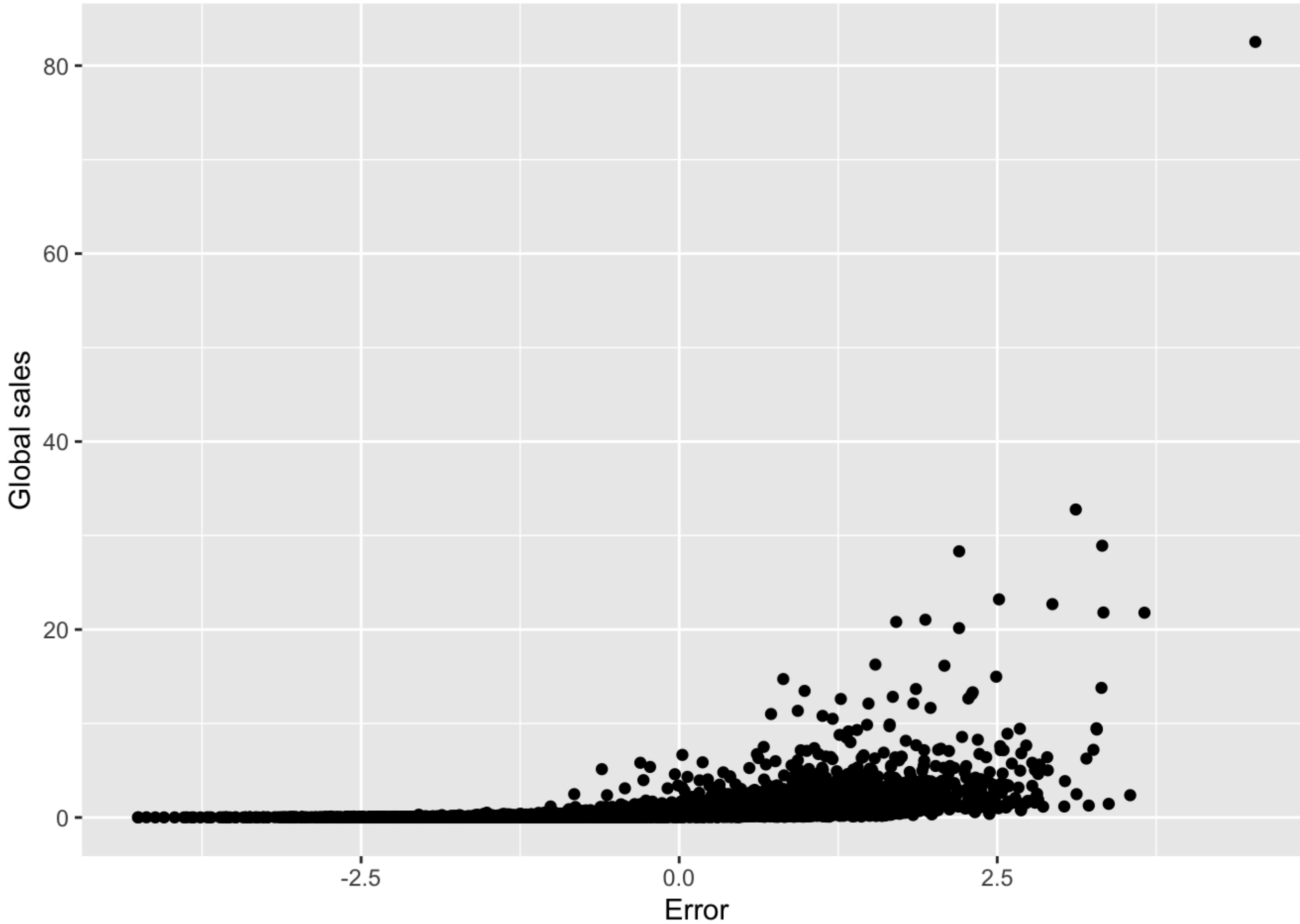
- Actual vs Predicted graph

```
ggplot(test_set) +
  geom_point(aes(log(Global_Sales), predicted_l1)) +
  geom_line(aes(log(Global_Sales), log(Global_Sales))) +
  xlab("Actual values") + ylab("Predicted values")+
  ggtitle("Actual Values vs Predicted Values")
```



- Error vs Sales

```
ggplot(test_set) + geom_point(aes(log(Global_Sales) - predicted_l1, Global_Sales)) +
  xlab("Error") + ylab("Global sales")
```



L2 - Ridge Model

- Ridge regression is essentially regularized linear regression
- Instead of getting rid of features that do not contribute to the model, ridge regression minimizes its impact on the trained model
- Ridge keeps all the features but is only significantly impacted by the most important features

```
model_l2 <- train(log(Global_Sales) ~ Critic_Score +
                    User_Score + Genre +
                    Year_of_Release +  Critic_Count +
                    User_Count + Rating +
                    publisher_top + developer_top +
                    num_of_platforms, method = "ridge", data = train_set)

# predicted values and RMSE
test_set$predicted_l2 <- predict(model_l2, test_set)
rmse_results <- rmse_results %>% add_row (Method = "L2 Ridge",
                           RMSE = RMSE(log(test_set$Global_Sales), test_set$predicted_l
2))
```

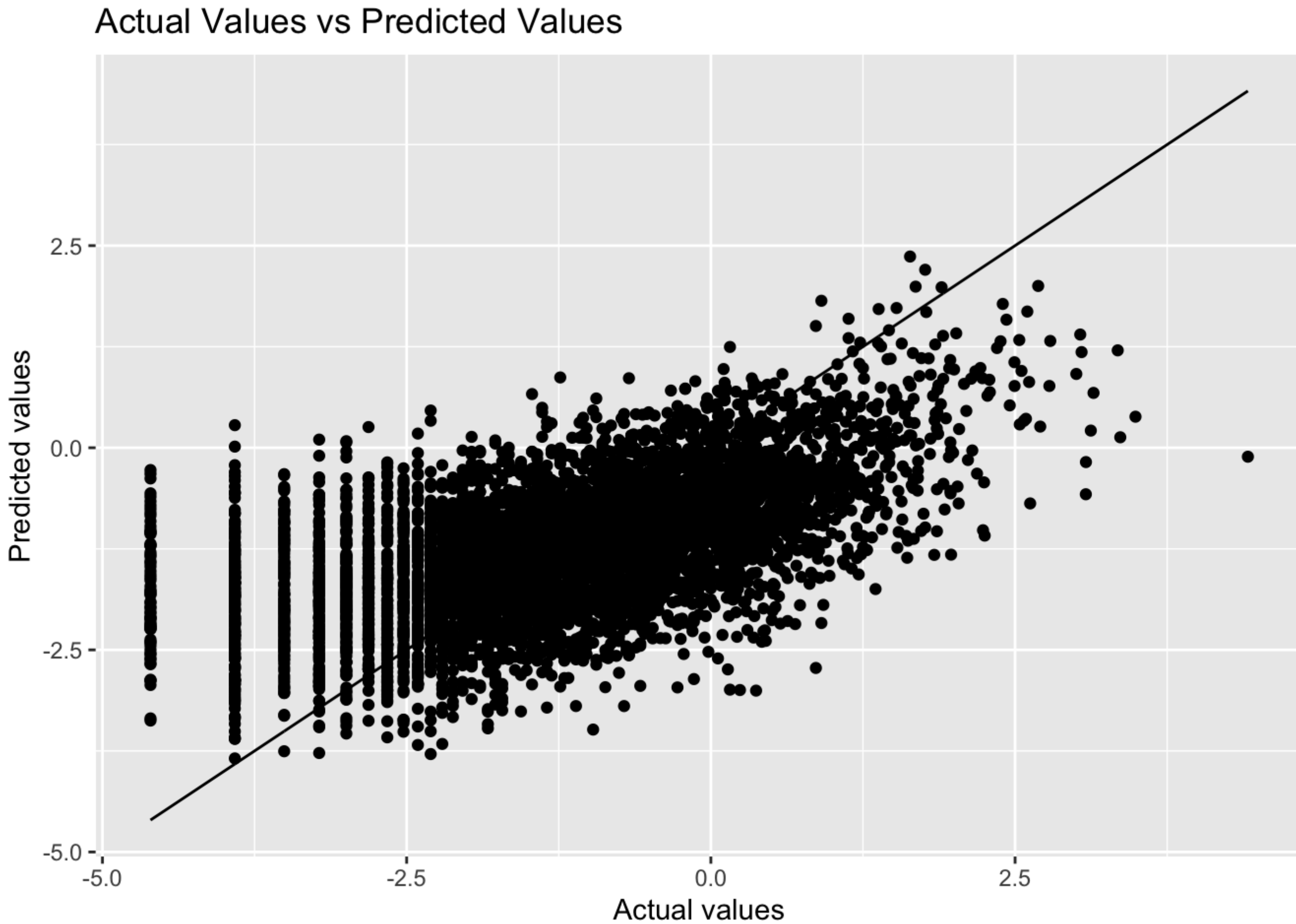
- L2 Model Summary

```
summary(model_l2)
```

```
##              Length Class      Mode
## call           4    -none-     call
## actions       25    -none-     list
## allset        22    -none-     numeric
## beta.pure    550    -none-     numeric
## vn            22    -none-     character
## mu             1    -none-     numeric
## normx         22    -none-     numeric
## meanx         22    -none-     numeric
## lambda         1    -none-     numeric
## L1norm        25    -none-     numeric
## penalty       25    -none-     numeric
## df            25    -none-     numeric
## Cp            25    -none-     numeric
## sigma2         1    -none-     numeric
## xNames        22    -none-     character
## problemType    1    -none-     character
## tuneValue      1    data.frame list
## obsLevels      1    -none-     logical
## param          0    -none-     list
```

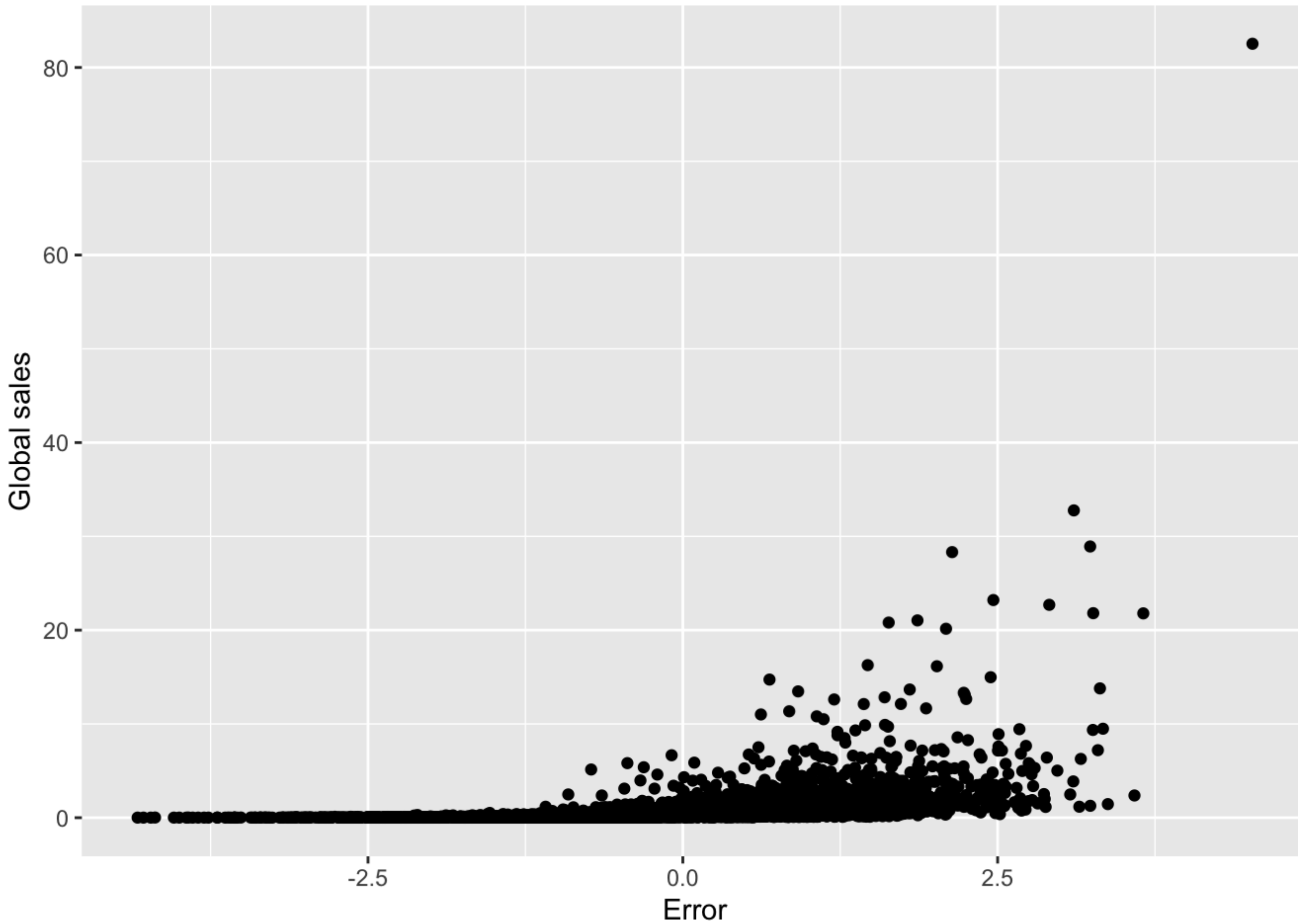
- Errors vs Predicted Plot

```
ggplot(test_set) +
  geom_point(aes(log(Global_Sales), predicted_l2)) +
  geom_line(aes(log(Global_Sales), log(Global_Sales))) +
  xlab("Actual values") + ylab("Predicted values")+
  ggtitle("Actual Values vs Predicted Values")
```



- Error vs Sales Plot

```
ggplot(test_set) + geom_point(aes(log(Global_Sales) - predicted_l2, Global_Sales)) +
  xlab("Error") + ylab("Global sales")
```



Random Forest Model

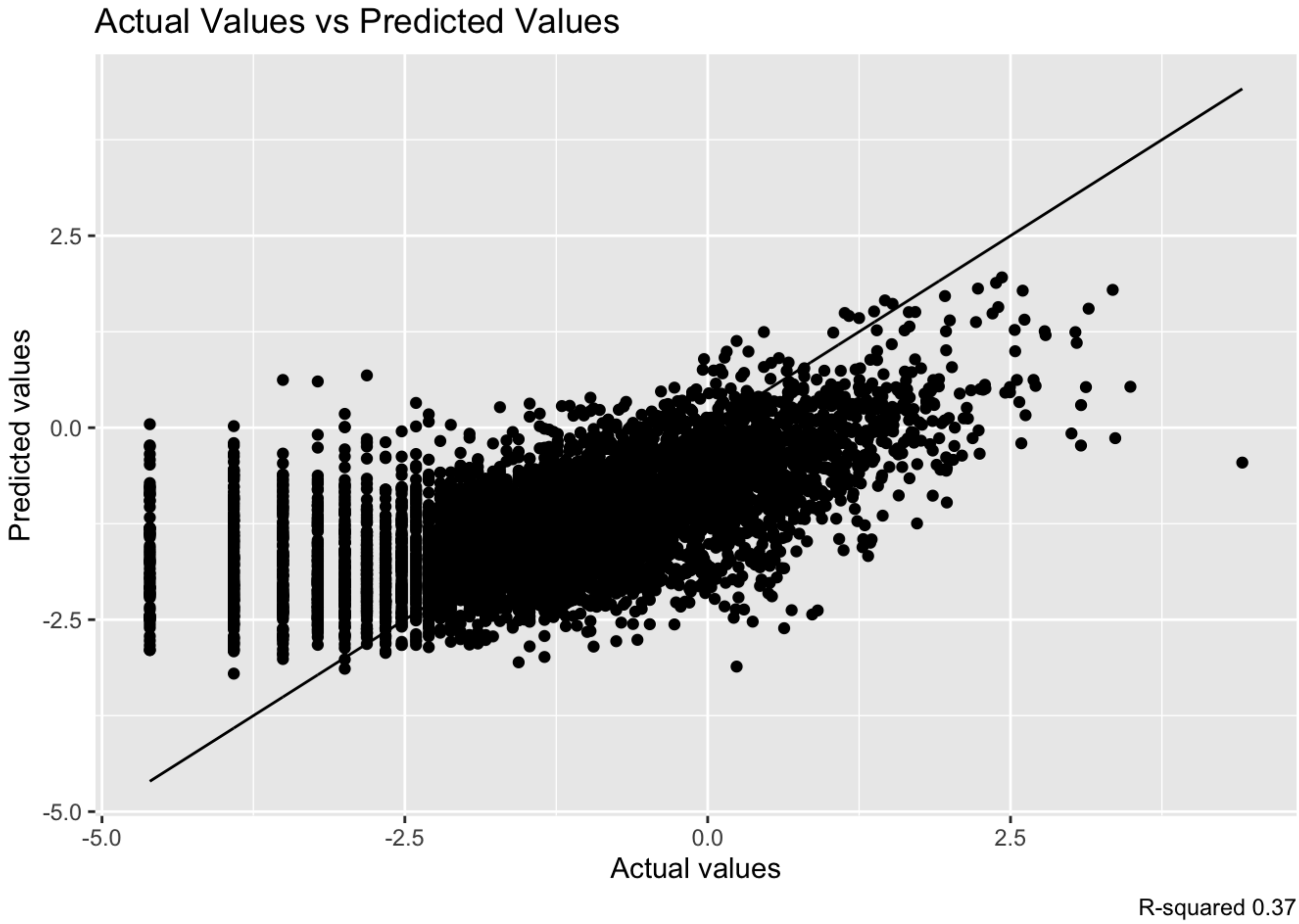
- This model will take a few minutes to run because there are multiple decision trees which can make the algorithm slow
- Using trainControl() because it helps specify a particular number of parameters
- Within trainControl(), the method = repeatedcv because the parameters will repeat accordingly
- splitrule = extratrees,variance because extratrees helps us specify and variance is used because it is the default typically
- Using method = ranger because it is performing recursive partitioning aka fast implementation of random forests

```
cntrl <- trainControl(method = "repeatedcv", number = 10,
                      repeats = 3)
tunegrid <- expand.grid(.mtry=c(1:5),
                        .min.node.size = seq(1, 5, 1),
                        .splitrule = c("extratrees", "variance"))
model_rf <- train(log(Global_Sales) ~ Critic_Score +
                    User_Score + Genre +
                    Year_of_Release + Critic_Count +
                    User_Count + Rating +
                    publisher_top + developer_top +
                    num_of_platforms, data = train_set,
                  method = "ranger", trControl = cntrl,
                  tuneGrid = tunegrid)

# predicted and RMSE
test_set$predicted_rf <- predict(model_rf, test_set)
rmse_results <- rmse_results %>% add_row(Method = "Random Forest",
                RMSE = RMSE(log(test_set$Global_Sales), test_set$predicted_rf))
```

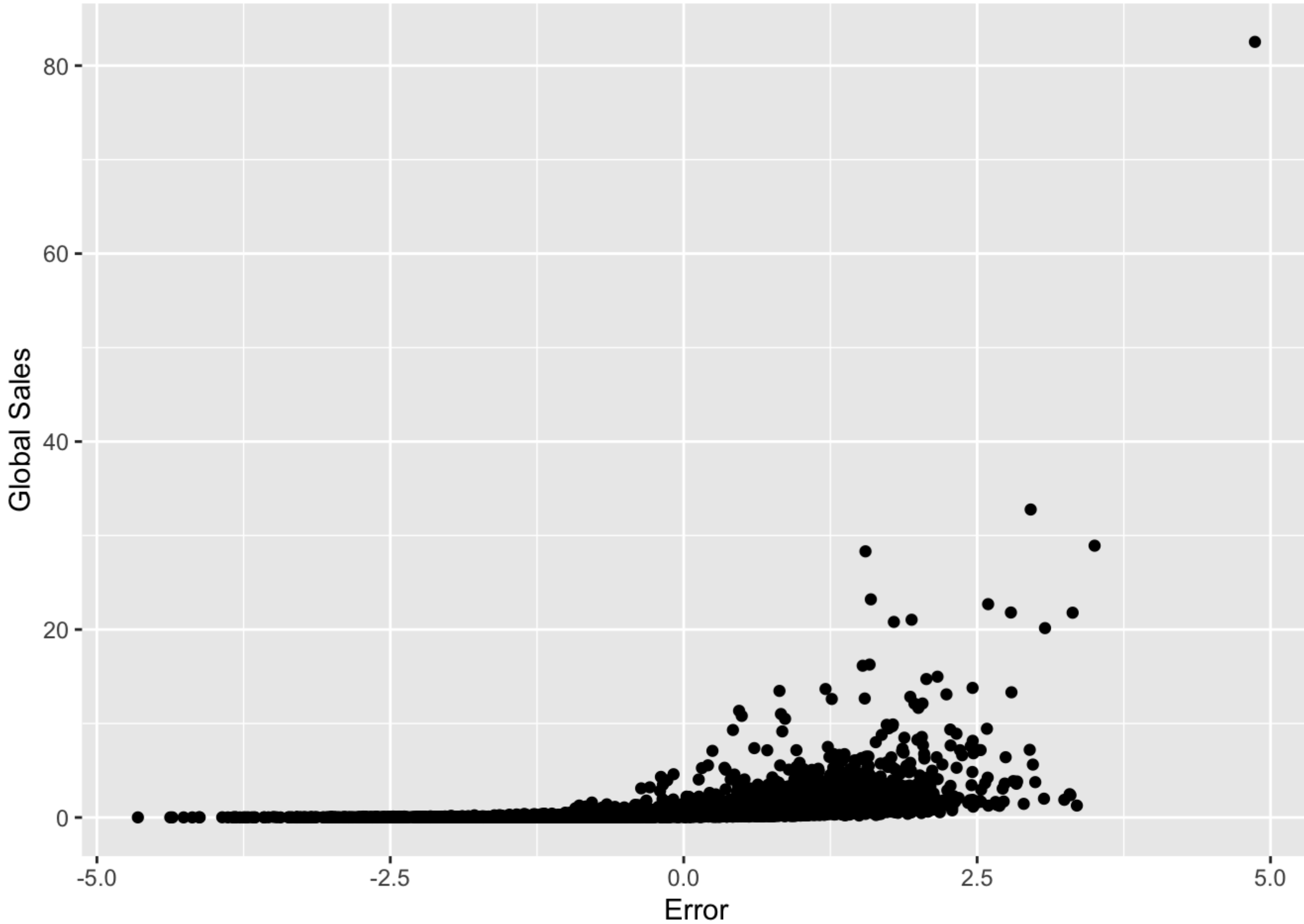
- Actual vs Predicted

```
ggplot(test_set) +
  geom_point(aes(log(Global_Sales), predicted_rf)) +
  geom_line(aes(log(Global_Sales), log(Global_Sales))) +
  xlab("Actual values") + ylab("Predicted values") +
  labs(caption =
         paste("R-squared",
               format(model_rf$finalModel$r.squared,
                      digits = 2)))+
  ggtitle("Actual Values vs Predicted Values")
```



- Error vs Global Sales

```
ggplot(test_set) + geom_point(aes(log(Global_Sales) - predicted_rf, Global_Sales)) +
  xlab("Error") + ylab("Global Sales")
```



Test That the RMSE is less than 2

- The reason why I am setting the limit for RMSE to 2 is because we want the RMSE value to be as low as possible, in general the models' RMSE were rarely going over 2 so I set that limit based on patterns that I saw

```
# testing linear regression RMSE
linear_regression_test <- rmse_results[1,2]

test_that("double",{
  expect_lt(linear_regression_test,2)
})
```

```
## Test passed 🎉
```

```
# testing SVM linear RMSE
SVM_Linear_test <- rmse_results[2,2]

test_that("double",{
  expect_lt(SVM_Linear_test,2)
})
```

```
## Test passed 🌈
```

```
# testing SVM poly RMSE
SVM_Polynomial_test <- rmse_results[3,2]
test_that("double",{
  expect_lt(SVM_Polynomial_test,2)
})
```

```
## Test passed 😀
```

```
# testing SVM radial RMSE
SVM_Radial_test <- rmse_results[4,2]
test_that("double",{
  expect_lt(SVM_Radial_test,2)
})
```

```
## Test passed 🥳
```

```
# testing L1 RMSE
L1_test <- rmse_results[5,2]
test_that("double",{
  expect_lt(L1_test,2)
})
```

```
## Test passed 🎊
```

```
# testing L2 RMSE
L2_test <- rmse_results[6,2]
test_that("double",{
  expect_lt(L2_test,2)
})
```

```
## Test passed 😸
```

```
# testing random forest RMSE
random_forest_test <- rmse_results[7,2]
test_that("double",{
  expect_lt(random_forest_test,2)
})
```

```
## Test passed 🎉
```

- Comparing the RMSE values of each model

```
print(rmse_results)
```

```
##               Method     RMSE
## 1 Linear Regression 1.140914
## 2        SVM Linear 1.147282
## 3    SVM Polynomial 1.136365
## 4        SVM Radial 1.140746
## 5          L1 Lasso 1.137891
## 6          L2 Ridge 1.140906
## 7     Random Forest 1.084544
```

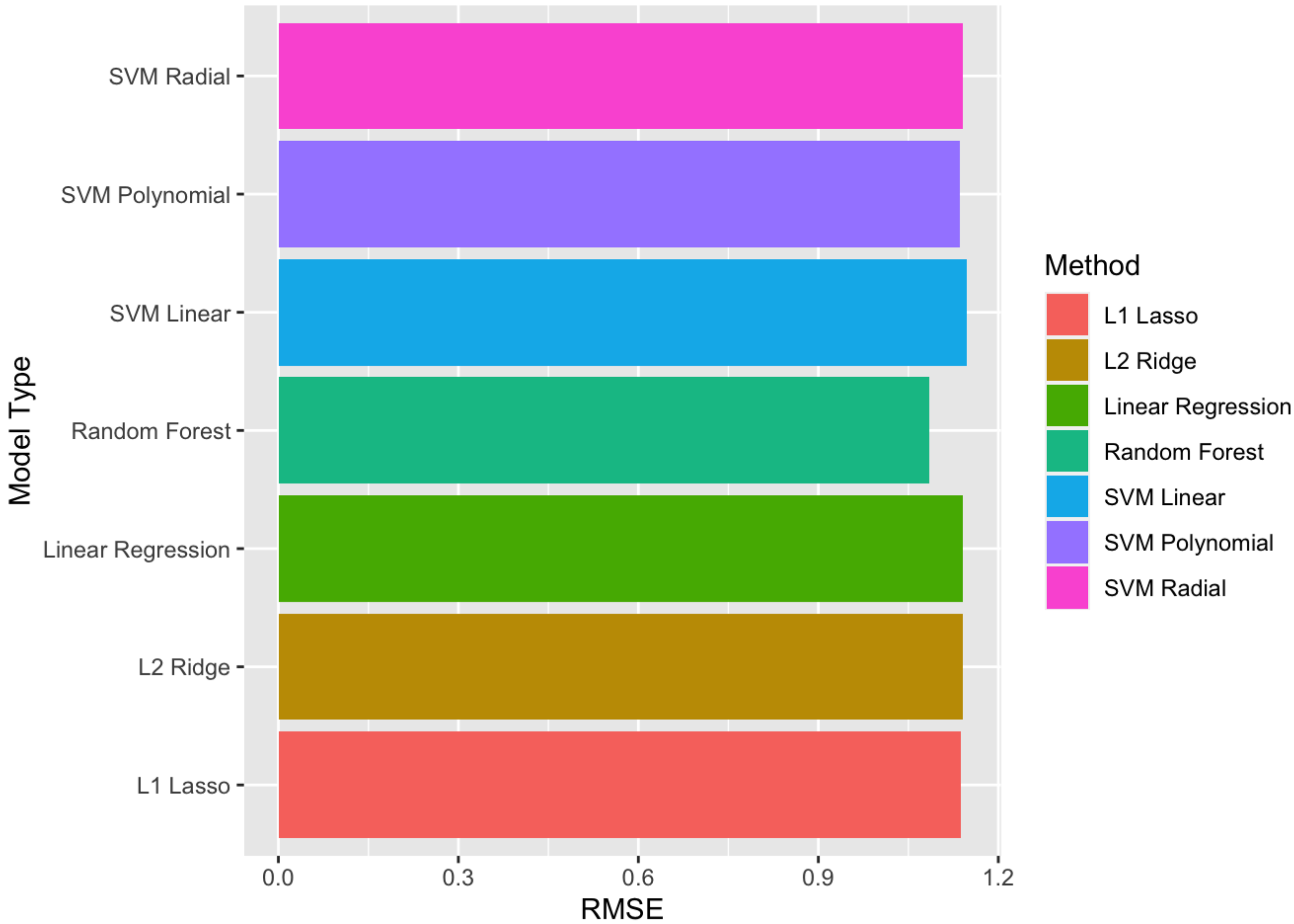
- Plotting and comparing all the models RMSE's
- Random forest did best!
- Note that the lower the RMSE, the better the fit

```
rmse_plot <- ggplot(rmse_results, aes(x=RMSE,y=Method, fill =Method))+
  geom_bar(stat="identity")+
  xlab("RMSE") + ylab("Model Type")

theme(text = element_text(size=10),
      legend.position="right",
      axis.text.x=element_text(angle = 90,vjust = 0.5,hjust = 1,size=8))
```

```
## List of 3
##  $ text           :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : num 10
##   ..$ hjust        : NULL
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : NULL
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi FALSE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ axis.text.x    :List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : num 8
##   ..$ hjust        : num 1
##   ..$ vjust        : num 0.5
##   ..$ angle        : num 90
##   ..$ lineheight   : NULL
##   ..$ margin       : NULL
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi FALSE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  $ legend.position: chr "right"
##  - attr(*, "class")= chr [1:2] "theme" "gg"
##  - attr(*, "complete")= logi FALSE
##  - attr(*, "validate")= logi TRUE
```

```
rmse_plot
```


SVM Radial
SVM Polynomial
SVM Linear
Random Forest
Linear Regression
L2 Ridge
L1 Lasso
Model Type
0.0
0.3
0.6
0.9
1.2
RMSE
Method
L1 Lasso
L2 Ridge
Linear Regression
Random Forest
SVM Linear
SVM Polynomial
SVM Radial