Predicting Sales of Video Games

Nabeel Arif

21/06/2020

Introduction

Video games are a multi-billion dollar industry with vast sales across the various regions in the world. The aim of this project to predict sales in North America based on historical sales data.

The data used for this analysis was obtained from Kaggle via the below link:

https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings/data

We will clean the data ready for exploration and visualization and then create models to predict North American sales. How well the models perform will be measured by the residual mean squared error (RMSE) and we will try various techniques and models to get this as low as possible.

Data Pre-Processing

Before we can use or even visualize the data we need to manipulate it into a format to do so. An overview of the data shows that the data is already spread into rows and columns:

```
# view raw data file
data %>% as_tibble()
```

```
## # A tibble: 16,719 x 16
##    Name  Platform Year_of_Release Genre Publisher NA_Sales EU_Sales JP_Sales
##    <fct> <fct>    <fct>           <fct> <fct>        <dbl>    <dbl>    <dbl>
##  1 Wii ~ Wii      2006            Spor~ Nintendo      41.4    29.0      3.77
##  2 Supe~ NES      1985            Plat~ Nintendo      29.1     3.58     6.81
##  3 Mari~ Wii      2008            Raci~ Nintendo      15.7    12.8      3.79
##  4 Wii ~ Wii      2009            Spor~ Nintendo      15.6    10.9      3.28
##  5 Poke~ GB       1996            Role~ Nintendo      11.3     8.89    10.2
##  6 Tetr~ GB       1989            Puzz~ Nintendo      23.2     2.26     4.22
##  7 New ~ DS       2006            Plat~ Nintendo      11.3     9.14     6.5
##  8 Wii ~ Wii      2006            Misc  Nintendo      14.0     9.18     2.93
##  9 New ~ Wii      2009            Plat~ Nintendo      14.4     6.94     4.7
## 10 Duck~ NES      1984            Shoo~ Nintendo      26.9     0.63     0.28
## # ... with 16,709 more rows, and 8 more variables: Other_Sales <dbl>,
## #   Global_Sales <dbl>, Critic_Score <int>, Critic_Count <int>,
## #   User_Score <fct>, User_Count <int>, Developer <fct>, Rating <fct>
```

The downloaded dataset has over 16000 rows and 16 columns, however as we will see some of this is not useable. Firstly the dataset details from Kaggle provide information on the data that it contains data up to November 2016 so any rows in 'Year_of_Release' post 2016 can be ignored.

```
# filter out rows post 2016
data <- data %>% filter((as.numeric(as.character(data$Year_of_Release)))<=2016)
```

The User_Score column values are stored as factors which is incorrect and we need to update these to numeric.

```
# Update User_Score column to numeric
data$User_Score <- as.numeric(as.character(data$User_Score))
```

Secondly there are a number of rows where data is missing (NA) or blank. This causes an issue as the model features require this in order to generate predictions. We can either estimate these, for example we could find the mean of the Critic_Score and use this value to fill in the missing values. However, in this case that would not be suitable as reviews or scores are subjective, what one person likes another may not. Further, for critical scores there is a count of critics who provided a score, so each score carries a weight and just using the mean is too simplistic.

Note however that for sales figures some rows have a zero value. This is not missing data rather actually indicates no sales and therefore this data will remain.

```
# overview of columns with NA
colSums(is.na(data))
```

```
##            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
##            8463            8463            8983            8983               0
##          Rating
##               0
```

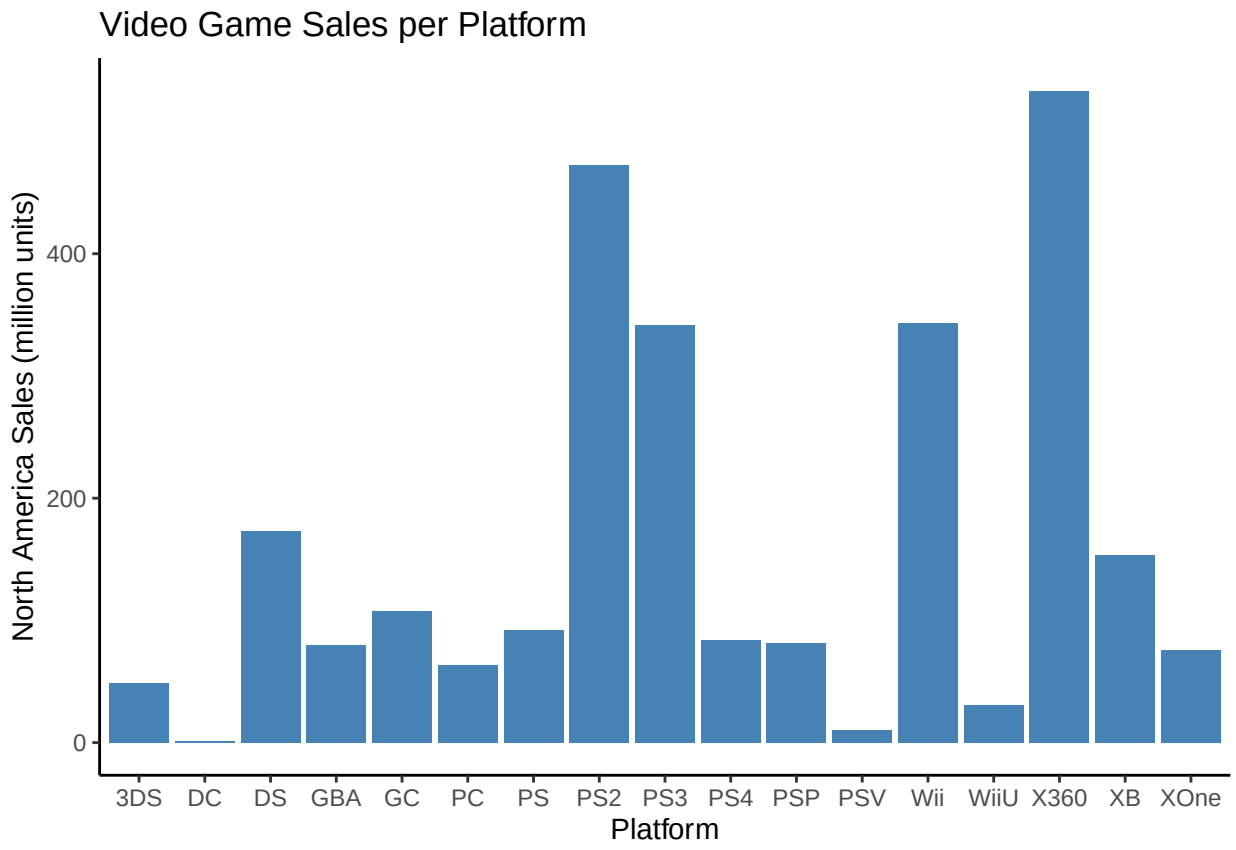
```
# remove rows that have any NA values
data <- na.omit(data)

# overview of columns with blanks
colSums(data == '')
```

```
##            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               4
##          Rating
##              68
```

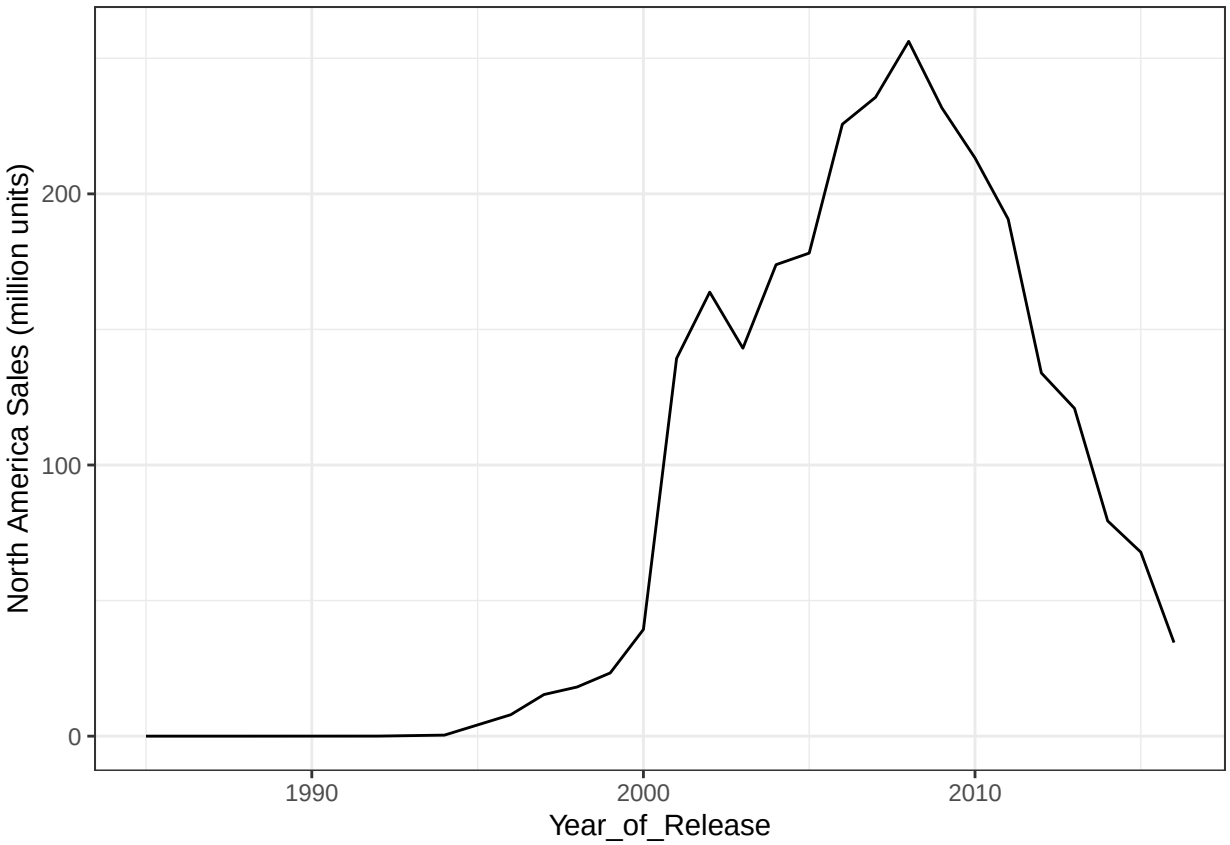
```
# remove rows with blank data
game_sales <- data %>% filter(Developer != '' & Rating != '')
```

Data Exploration and Visualization

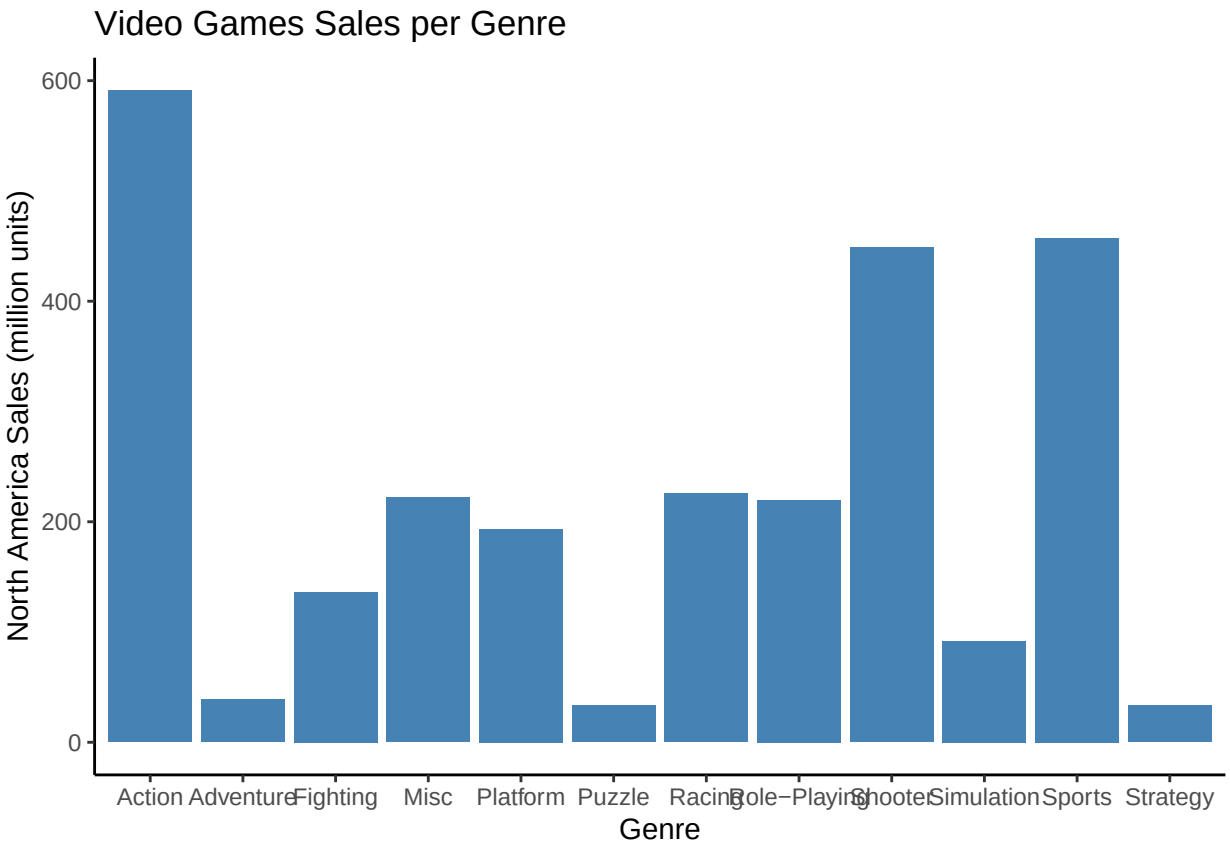
Now we have the data in a clean format we can explore and visualize and see if we can identify any patterns. Firstly, we'll look at the various platforms available for video games and see how sales figures vary.

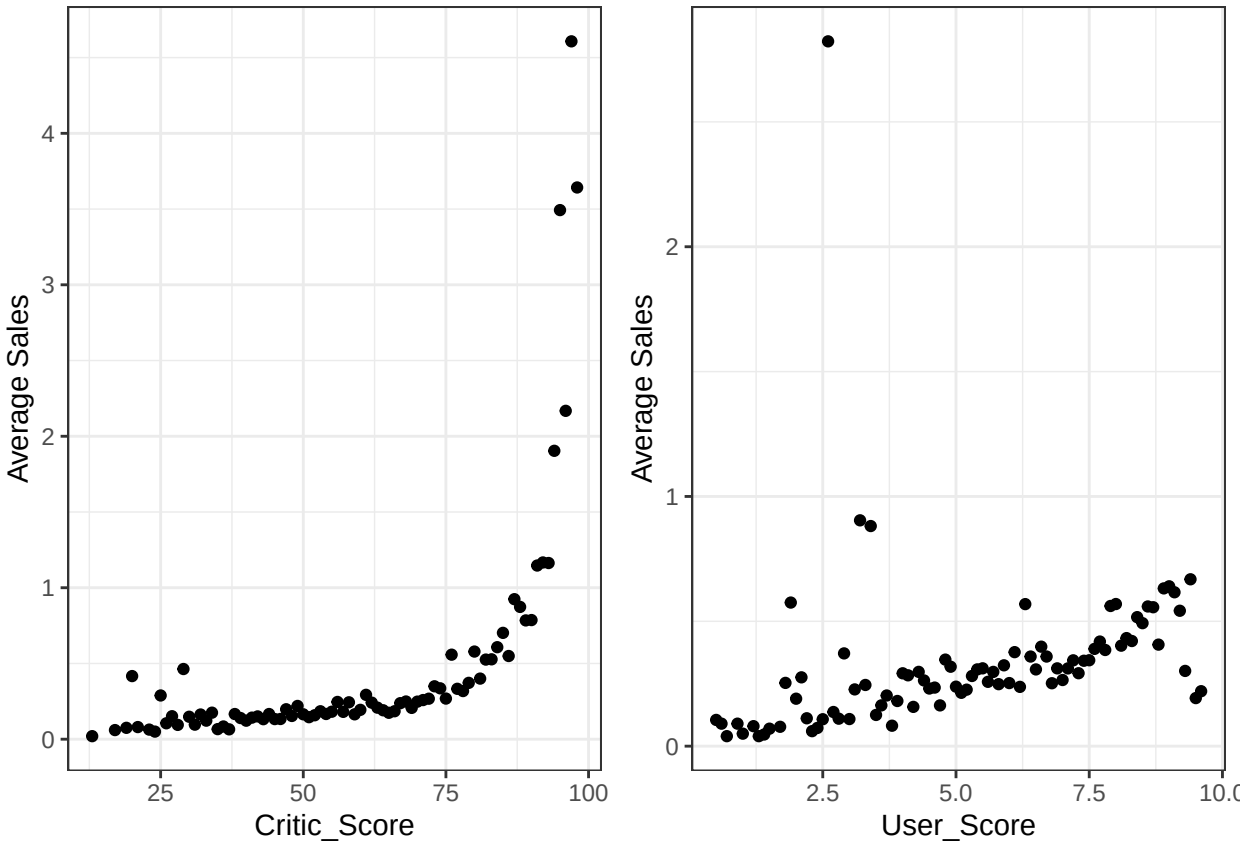


We can see that some platforms are more popular than others. Next, we'll see how sales figures have changed over time.



Let us continue to make plots for sales against genre, critic score and user score.


Video Games Sales per Genre
North America Sales (million units)
600
400
200
0
Action
Adventure
Fighting
Misc
Platform
Puzzle
Racing
Role−Playing
Shooter
Simulation
Sports
Strategy
Genre



We can see a positive relationship, almost linear relationship between critic and user score and sales with very high critic scores relating to very high sales that do not fall within that linear relationship. We can also look at the "best" games, those rated the highest by critics.

```
##                         Name Platform NA_Sales Critic_Score Critic_Count
## 1       Grand Theft Auto IV     X360     6.76           98           86
## 2       Grand Theft Auto IV      PS3     4.76           98           64
## 3  Tony Hawk's Pro Skater 2       PS     3.05           98           19
## 4                SoulCalibur       DC     0.00           98           24
## 5        Grand Theft Auto V      PS3     7.02           97           50
## 6        Grand Theft Auto V     X360     9.66           97           58
## 7      Grand Theft Auto III      PS2     6.99           97           56
## 8        Grand Theft Auto V      PS4     3.96           97           66
## 9        Super Mario Galaxy      Wii     6.06           97           73
## 10     Super Mario Galaxy 2      Wii     3.56           97           87
```

From this it's' quite evident that the same game i.e. same name, but on different platforms can appear so let's take that into account and group by name, and take the average critic score and total number of critics for the same games. However, note that not all games are available across all platforms.

```
## # A tibble: 10 x 4
##    Name                                NA_Sales Avg_Critic_Score Total_Critics
##    <fct>                                  <dbl>            <dbl>         <int>
##  1 SoulCalibur                             0                98            24
##  2 Metroid Prime                           1.96             97            70
##  3 NFL 2K1                                 1.02             97            22
```

```
##  4 Super Mario Galaxy                     6.06            97             73
##  5 Super Mario Galaxy 2                   3.56            97             87
##  6 Grand Theft Auto V                    23.8             96.8          245
##  7 Tony Hawk's Pro Skater 2               3.9             96.5           35
##  8 Gran Turismo                           4.02            96             16
##  9 Metal Gear Solid 2: Sons of Liberty    2.45            96             42
## 10 Tekken 3                               3.27            96             15
```

Similarly we can see the "worst" games with platform,

```
##                                 Name Platform NA_Sales Critic_Score
## 1                        Ride to Hell      PS3     0.02           13
## 2  Leisure Suit Larry: Box Office Bust      PS3     0.06           17
## 3              Nickelodeon Party Blast       XB     0.16           19
## 4                         Chicken Shoot       DS     0.13           19
## 5                              Rugby 15      PS4     0.04           19
## 6                            Anubis II      Wii     0.06           19
## 7                        Ride to Hell     X360     0.04           19
## 8                        Balls of Fury      Wii     0.02           19
## 9                     Deal or No Deal       DS     1.15           20
## 10                             Rugby 15     XOne     0.03           20
##    Critic_Count
## 1             4
## 2            11
## 3             4
## 4             4
## 5             5
## 6             5
## 7            14
## 8             6
## 9            13
## 10            4
```

and grouping games with the same name.

```
## # A tibble: 10 x 4
##    Name                               NA_Sales Avg_Critic_Score Total_Critics
##    <fct>                                 <dbl>            <dbl>         <int>
##  1 Ride to Hell                           0.06             16              18
##  2 Anubis II                              0.06             19               5
##  3 Balls of Fury                          0.02             19               6
##  4 Nickelodeon Party Blast                0.16             19               4
##  5 Rugby 15                               0.07             19.5             9
##  6 Ninjabread Man                         0.07             20               6
##  7 Leisure Suit Larry: Box Office Bust    0.2              21              38
##  8 Self-Defense Training Camp             0.08             21               5
##  9 Charlie's Angels                       0.01             23              21
## 10 Chicken Shoot                          0.45             23              13
```

We have not shown it here but we can also obtain the same results for "best" and "worst" games according to User_Score.

Lastly let's' see if we can find any correlation between the numeric fields; NA_Sales, Critic_Score, Critic_Count, User_Score and User_Count.

```r
# correlation between features by rank
corr_matrix <- cor(game_sales[, c(6, 11:14)], method = "spearman")
corr_matrix
```

```
##               NA_Sales Critic_Score Critic_Count User_Score User_Count
## NA_Sales     1.0000000    0.3141524    0.3360029  0.1282123  0.2761658
## Critic_Score 0.3141524    1.0000000    0.3959633  0.5359020  0.4877669
## Critic_Count 0.3360029    0.3959633    1.0000000  0.1970988  0.5832490
## User_Score   0.1282123    0.5359020    0.1970988  1.0000000  0.1664156
## User_Count   0.2761658    0.4877669    0.5832490  0.1664156  1.0000000
```

We can see the positive relationship between scores and sales discussed earlier.

Methods & Models

Naive Model

Before we build a model, we need to split the data as we will train the models on a training set of the data and predict sales figures on a test set. This is split as 80% for training and 20% for testing respectively. This split has been chosen as we will have several features so need a larger validation set to test this on.

```r
# split data into training and test sets
test_index <- createDataPartition(y = game_sales$NA_Sales, times = 1, p = 0.2, list = FALSE)

# define training set
train_set <- game_sales[-test_index,]

# define test set
test_set <- game_sales[test_index,]
```

The models will need to be compared in terms of effectiveness so we will use the RMSE to do so. This is defined as below

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

We define $y_{g,c}$ as the sales figure for game g and critic score c and $\hat{y}_{g,c}$ our estimate with the best model having the lowest RMSE value.

Now we can build a naive model which will serve as a benchmark for all models. We can define this model as:

$$Y_{g,c} = \mu + \varepsilon_{g,c}$$

Where $Y_{g,c}$ is the predicted sales of game g and critic score c and $\varepsilon_{c,g}$ independent errors and μ the average sales for all games. We know that the estimate that minimizes the RMSE is the least squares estimate of μ which is just the average.

```r
# average NA sales
mu_hat <- mean(train_set$NA_Sales)
mu_hat
```

```
## [1] 0.3932094
```

We can use this to calculate RMSE.

```r
# calculate RMSE for naive model
naive_rmse <- RMSE(test_set$NA_Sales, mu_hat)
```

Model	Predicted_RMSE
Naive	**0.8624597**

The RMSE for a naive approach is under one which is a good sign, however, now that we have established a benchmark, we can try other models to reduce this.

Linear Regression Model

From the data we can see that some games are scored higher than others. We can therefore add an extra term to our model, b_c, to represent the score given. (Note that each row in the data is unique in terms of, one game on a certain platform, can only have one critic score)

$$Y_{g,c} = \mu + b_c + \varepsilon_{g,c}$$

Just like there is consideration taken for the critic score, we also need to consider user score b_u, platform b_p, genre b_a, critic count b_d and user count b_e. So our model actually becomes:

$$Y_{g,c} = \mu + b_c + b_u + b_p + b_a + b_d + b_e + \varepsilon$$

```r
# build linear regression model on training set
lr_model <- train(NA_Sales ~ Critic_Score + User_Score + Platform + Genre +
                    Critic_Count + User_Count,
                  data=train_set,
                  method="lm")

# generate predictions on test set
y_hat <- predict(lr_model, test_set)

# calculate RMSE for linear regression model
lr_rmse<- RMSE(y_hat, test_set$NA_Sales)
```

Model	Predicted_RMSE
Naive	0.8624597
Linear Regression	**0.7482631**

We have managed to reduce our RMSE, however remember from the critic score against average sales plot we saw that for very high critic scores the resulting sales were also very high and did not follow a linear relationship. We'll continue to explore other models to see if we can improve.

Elastic Net Model

From the data we can see that the "best" and "worst" scored games can have a low count of critics. This leads to large estimates of b_c which can increase RMSE. Regularization permits us to penalize large estimates that are formed using small sample sizes.

Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

$$\frac{1}{N}\sum_{g,c}(y_{g,c}-\mu-b_c)^2+\lambda\sum_c b_c^2$$

This approach will have our desired effect: when our sample size n_c is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_c+\lambda \approx n_c$. However, when the n_c is small, then the estimate $\hat{b}_c(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink. Note however that λ is a tuning parameter and we can use cross-validation to choose it.

We can use regularization for the other effects also and penalize large estimates. The above formula is for a L2 Regularization where the penalty term, b, is squared whereas as L1 regularization uses the absolute value of b. Elastic Net models utilises both and introduces another parameter α which again can be tuned and we will use this for our model.

```
# Set training control
train_control <- trainControl(method = "repeatedcv",
                              number = 5,
                              repeats = 5,
                              search = "random",
                              verboseIter = FALSE)

# train elastic net model on training data
enet_model <- train(NA_Sales ~ Critic_Score + User_Score + Platform + Genre +
                      Critic_Count + User_Count,
                         data = train_set,
                         method = "glmnet",
                         tuneLength = 25,
                         trControl = train_control)

# generate predictions on test data
y_hat <- predict(enet_model, test_set)

# calculate RMSE for Elastic Net model
enet_rmse<- RMSE(y_hat, test_set$NA_Sales)
```

Model	Predicted_RMSE
Naive	0.8624597
Linear Regression	0.7482631
Elastic Net	**0.7482408**

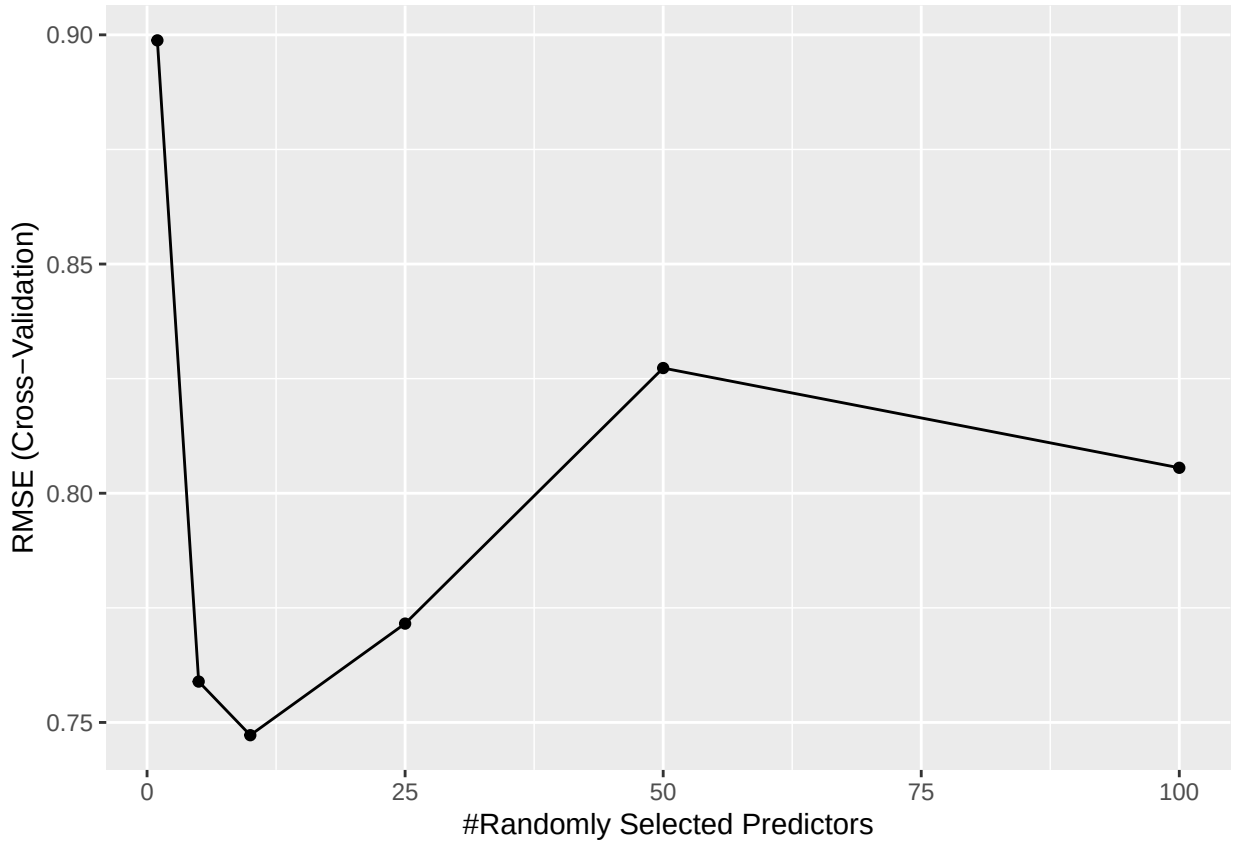
We have reduced RMSE but only incrementally. Let's take a different path to try and improve our model.

Random Forest

Our last model will use a Random Forest which are used to improve predictions by generating many predictors, each using regression or classification trees, and then forming a final prediction based on the average

prediction of all these trees. We will find the optimal model changing the parameter that controls the minimum number of data points in the nodes of the tree.

We can see the results of the tuning and how the error is affected by changes in parameters.



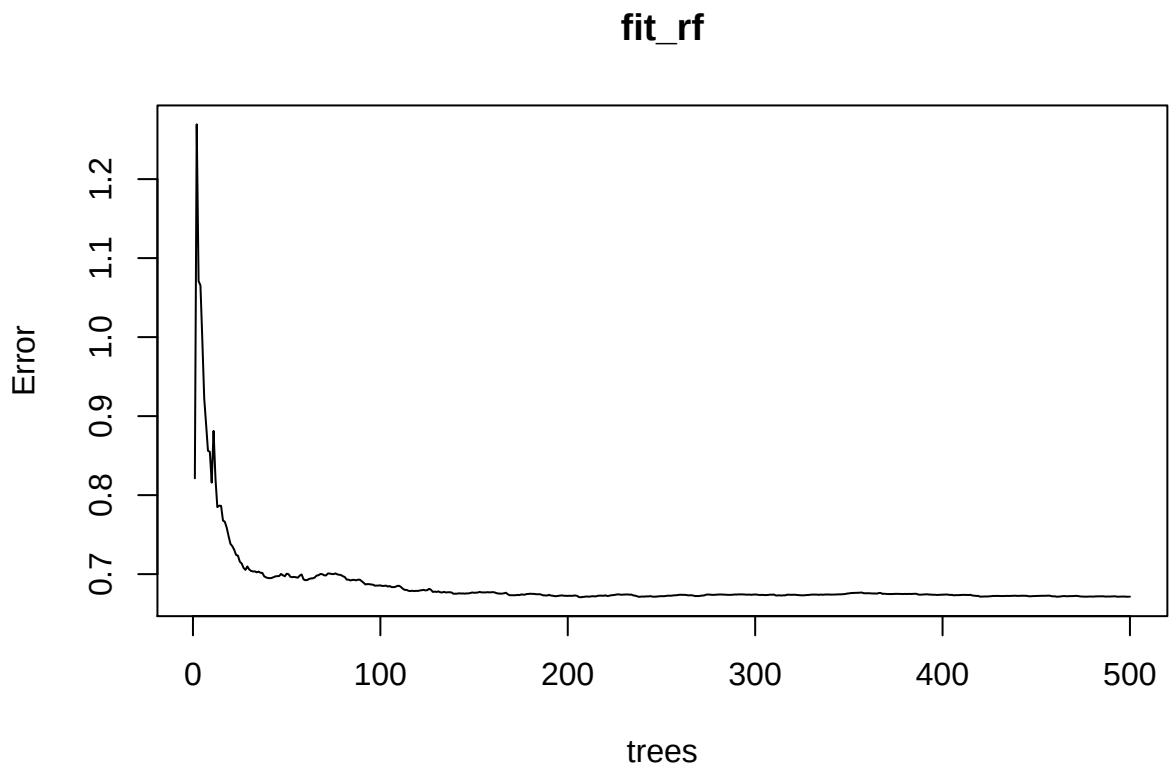
We can then obtain the best value for our parameter and fit the final model.

```
# get parameter value that is best tune for model
train_rf$bestTune
```

```
##   mtry
## 3   10
```

```
# fit the model using optimized nodes
fit_rf <- randomForest(NA_Sales ~ Critic_Score + User_Score + Platform + Genre +
                         Critic_Count + User_Count,
                       data = train_set,
                       minNode = train_rf$bestTune$mtry)
```

We check to ensure we have used enough trees to minimize error.



The number of trees looks correct so we can now make our predictions and calculate RMSE.

```
# generate predictions on test data
y_hat <- predict(fit_rf, test_set)

# calculate RMSE for Random Forest model
rf_rmse <- RMSE(y_hat, test_set$NA_Sales)
```

Model	Predicted_RMSE
Naive	0.8624597
Linear Regression	0.7482631
Elastic Net	0.7482408
Random Forest	**0.6337469**

The result from the Random Forest does reduce RMSE further and in fact provides the lowest RMSE of all models.

Results

Below is a summary of the models used and their respective RMSE.

Model	**Predicted_RMSE**
Naive	**0.8624597**
Linear Regression	**0.7482631**
Elastic Net	**0.7482408**
Random Forest	**0.6337469**

We can see that a naive model yields the highest RMSE which is expected as it does not consider any feature bias. As we move through the models there is an incremental improvement with Random Forests providing the best results which is not surprising since a Random Forest model generates many predictors and then averages.

Conclusion

We have managed to build a model with a low RMSE when predicting North American sales of video games. This was done so by taking into account the different features and bias within the data and incorporating these into the models.

There are a few issues or limitations of these models. Firstly, the original data set from Kaggle had to be cleaned which resulted in more than half of the original data being removed, thus resulting in a smaller data set. This can lead to overfitting as we have less data to train with.

Another factor to consider is that critic and user scores are only received once a game has been released. So those features would need to be ignored if modelling pre-release.

Improvements to the model can include matrix factorization and looking at relationships within the data such as if a certain region has a strong affiliation with a particular platform or genre.