Video Game Sales

Neill Smith

2022-09-20

Introduction - Video Game Sales Project

Video games are a massive worldwide industry. In 2021 the video games industry saw $191 billion in global sales, thousands of development studios, and hundreds of publishers employing hundreds of thousands of people worldwide. The "Video Game Sales as at 22 Dec 2016" data set from Kaggle contains sales information for video games from 1980 through 2016 as well as several other characteristics of the games including the Developer, Publisher, Platform, Genre, and ESRB Rating. We will dive into this data, clean it up, tease out some valuable insights, build some models with it, and make some predictions. Let us get started.

Methodology & Analysis

We will use the attributes of our Video Game Sales data set to create two regression models, one linear and one random forest, that attempt to predict Global Sales. Models such as these could be used to green-light new projects, set appropriate levels of funding, and set appropriate sales targets. To evaluate our models we will use a root mean squared error (RMSE) loss function. We will use cross-validation to evaluate factors for use in each model and adjust tuning parameters. Our final result will be evaluated against a hold-out set and the performance of the two models will be compared.

Data Exploration, Cleaning, & Visualization

We begin by exploring each attribute in the data set in detail, making corrections where necessary to make more accurate predictions, and examine some underlying trends with insightful visualizations.

Exploration & Cleaning

We begin with a check of the table structure and a basic quality assessment for missing data.

```
## Classes 'data.table' and 'data.frame':   16719 obs. of  16 variables:
##  $ Name           : Factor w/ 11562 levels "'98 Koshien",..: 11076 9389 5613 11078 7392 9779 6767 110
##  $ Platform       : Factor w/ 31 levels "2600","3DO","3DS",..: 27 12 27 27 6 6 5 27 27 12 ...
##  $ Year_of_Release: int  2006 1985 2008 2009 1996 1989 2006 2006 2009 1984 ...
##  $ Genre          : Factor w/ 12 levels "Action","Adventure",..: 11 5 7 11 8 6 5 4 5 9 ...
##  $ Publisher      : Factor w/ 580 levels "10TACLE Studios",..: 362 362 362 362 362 362 362 362 362 36
##  $ 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     : num  8 NA 8.3 8 NA NA 8.5 6.6 8.4 NA ...
##  $ User_Count     : int  322 NA 709 192 NA NA 431 129 594 NA ...
##  $ Developer      : Factor w/ 1696 levels "10tacle Studios",..: 1021 NA 1021 1021 NA NA 1021 1021 10
##  $ Rating         : Factor w/ 8 levels "AO","E","E10+",..: 2 NA 2 2 NA NA 2 2 2 NA ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The data is organized into 16 variables:

1. Name - Name of the game.
2. Platform - Console on which the game is running e.g. Playstation, Xbox, PC, etc.
3. Year_of_Release - Year the game was released.
4. Genre - Game's category.
5. Publisher - The Publisher of the game.
6. NA_Sales - Game sales in North America (in millions of units).
7. EU_Sales - Game sales in the European Union (in millions of units).
8. JP_Sales - Game sales in Japan (in millions of units).
9. Other_Sales - Game sales in the rest of the world, i.e. Africa, Asia excluding Japan, Australia, Europe excluding the E.U. and South America (in millions of units).
10. Global_Sales - Total sales in the world (in millions of units).
11. Critic_Score - Aggregate score compiled by Metacritic.
12. Critic_Count - The number of critics used in the Critic_Score.
13. User_Score - Score by Metacritic's subscribers.
14. User_Count - The number of users who gave the User_Score.
15. Developer - Party responsible for creating the game.
16. Rating - The ESRB ratings.

	#_of_NAs
Name	2
Year_of_Release	269
Genre	2
Publisher	54
Critic_Score	8582
Critic_Count	8582
User_Score	9129
User_Count	9129
Developer	6623
Rating	6769

Checking for missing data we see mostly missing values for Scores and Counts, Developer, and Rating with a handful of missing release years, publishers, and even a couple missing names. Let us explore the blank names first.

Name	Developer	Publisher	Genre	Year_of_Release	Rating
NA	NA	Acclaim Entertainment	NA	1993	NA
NA	NA	Acclaim Entertainment	NA	1993	NA

Not a lot of information here, in addition to the missing names several other fields are also blank: Developer, Genre, and Rating. There are to many missing fields for this data to be useful and there are only 2 such occurrences so we will discard it.

Next we will check the missing Year of Release fields:

```
##  [1] 2006 1985 2008 2009 1996 1989 1984 2005 1999 2007 2010 2013 2004 1990 1988
## [16] 2002 2001 2011 1998 2015 2012 2014 1992 1997 1993 1994 1982 2016 2003 1986
## [31] 2000   NA 1995 1991 1981 1987 1980 1983 2020 2017
```

Name	Year_of_Release	Global_Sales
wwe Smackdown vs. Raw 2006	NA	3.00
Yu Yu Hakusho: Dark Tournament	NA	0.21
eJay Clubworld	NA	0.15
Zero: Tsukihami no Kamen	NA	0.08
Yoostar on MTV	NA	0.07
Yu-Gi-Oh! 5D's Wheelie Breakers (JP sales)	NA	0.02

Here we can see the list of unique years in the data set along side some examples of titles that are missing year of release values. One value jumps out here, *WWE Smackdown vs. Raw 2006,* because it is missing a release year but appears to be a title released on a annual basis with each year in the title itself. We can extract such years and use them to fill in some of our missing year data. Let's try to find more examples.

Name	Year_of_Release
Madden NFL 2004	NA
FIFA Soccer 2004	NA
wwe Smackdown vs. Raw 2006	NA
NASCAR Thunder 2003	NA
PES 2009: Pro Evolution Soccer	NA
Madden NFL 2002	NA
NFL GameDay 2003	NA
NBA Live 2003	NA
Tomb Raider (2013)	NA
All-Star Baseball 2005	NA
NBA Live 2003	NA
All-Star Baseball 2005	NA
Tour de France 2011	NA
Sega Rally 2006	NA
PDC World Championship Darts 2008	NA
Football Manager 2007	NA
PDC World Championship Darts 2008	NA

Here we can see a list of examples of other titles where the year is in the title itself or included in parenthesis to differentiate titles with the same name by their year of release such as *Tomb Raider (2013).* We will extract these year values in the title and use them as our year of release and discard the rest.

Some other years in our list that seem strange for a 2016 data set are years greater than 2017:

Name	Year_of_Release
Imagine: Makeup Artist	2020
Phantasy Star Online 2 Episode 4: Deluxe Package	2017
Phantasy Star Online 2 Episode 4: Deluxe Package	2017
Brothers Conflict: Precious Baby	2017

These may be pre-release listings for titles that were slated to come out at the time the data was gathered but had not yet released. We will discard them from our data set.

Next we examine missing Publishers:

Name
wwe Smackdown vs. Raw 2006
Shrek / Shrek 2 2-in-1 Gameboy Advance Video
Bentley's Hackpack
Nicktoons Collection: Game Boy Advance Video Volume 1
SpongeBob SquarePants: Game Boy Advance Video Volume 1
SpongeBob SquarePants: Game Boy Advance Video Volume 2
The Fairly Odd Parents: Game Boy Advance Video Volume 1
The Fairly Odd Parents: Game Boy Advance Video Volume 2
Cartoon Network Collection: Game Boy Advance Video Platinum Edition
Sonic X: Game Boy Advance Video Volume 1

Same of these appear to be legitimate games we may want to include but many are anthologies of Video Volumes or special edition releases of other titles and not useful for our model. We will replace missing Publishers with the Developer, assuming those titles to be self published, and discard the rest.

Examining User Scores and Critic Scores:

```
##  [1] 8.0  NA 8.3 8.5 6.6 8.4 8.6 7.7 6.3 7.4 8.2 9.0 7.9 8.1 8.7 7.1 3.4 5.3 4.8
## [20] 3.2 8.9 6.4 7.8 7.5 2.6 7.2 9.2 7.0 7.3 4.3 7.6 5.7 5.0 9.1 6.5 8.8 6.9 9.4
## [39] 6.8 6.1 6.7 5.4 4.0 4.9 4.5 9.3 6.2 4.2 6.0 3.7 4.1 5.8 5.6 5.5 4.4 4.6 5.9
## [58] 3.9 3.1 2.9 5.2 3.3 4.7 5.1 3.5 2.5 1.9 3.0 2.7 2.2 2.0 9.5 2.1 3.6 2.8 1.8
## [77] 3.8 0.0 1.6 9.6 2.4 1.7 1.1 0.3 1.5 0.7 1.2 2.3 0.5 1.3 0.2 0.6 1.4 0.9 1.0
## [96] 9.7
```

```
##  [1] 76 NA 82 80 89 58 87 91 61 97 95 77 88 83 94 93 85 86 98 96 90 84 73 74 78
## [26] 92 71 72 68 62 49 67 81 66 56 79 70 59 64 75 60 63 69 50 25 42 44 55 48 57
## [51] 29 47 65 54 20 53 37 38 33 52 30 32 43 45 51 40 46 39 34 41 36 31 27 35 26
## [76] 19 28 23 24 21 17 13
```

Critic Scores are on a 0-100 scale with only whole numbers while User Scores are 0-10 with 1 decimal place. So effectively the same scale if we multiply User Scores by 10. We will make the adjustment. Many Scores are also missing but we do not plan to use scores in our final models. Scores could probably give us some measure of predictive power however these scores are only available after a game has been finished and we are more interested in modeling predictions earlier in the process before these scores would be available.

Next we will examine the blank Developers in the data:

Name	Developer	Publisher
Super Mario Bros.	NA	Nintendo
Pokemon Red/Pokemon Blue	NA	Nintendo
Tetris	NA	Nintendo
Duck Hunt	NA	Nintendo
Nintendogs	NA	Nintendo

As with the entries with blank Publishers before these appear to be legitimate data points and some are quite famous. The original *Super Mario Bros.* for example was developed by in in-house Nintendo studio.

We will use the Publisher as a proxy for the Developer in this case and fill in the missing Developer values in with the same value as Publisher. This is not entirely accurate to life but is a close enough approximation for our model.

Let's check our counts for blank data again and see how much progress has been made:

	#_of_NAs
Critic_Score	8436
Critic_Count	8436
User_Score	8958
User_Count	8958
Rating	6652

All but Rating have been sufficiently cleaned. Looking at a list of unique ratings:

Rating	Count
AO	1
E	3930
E10+	1394
EC	8
K-A	3
M	1537
RP	1
T	2905
NA	6652

We can see these match ESRB ratings, a common industry standard that has been in use for decades. Performing some further research on each of these ratings values yields some additional information:

Symbols	Rating	Years Active
EC	Early Childhood	1994-2018 Rolled into E
K-A	Kids to Adults	1994-1998 Rolled into E
T	Teen	1994-present
M	Mature	1994-present
AO	Adults Only	1994-present
E	Everyone	1998-present
E10+	Everyone 10 and over	2005-present
RP	Rating Pending	1994-present

Several of our ratings categories have very few data points and we can see from the history of these ratings that K-A and EC could easily be rolled into E as they were for the official ratings in 1998 and 2018 respectively. For AO and RP rated titles we have very few data points, examing further:

Name	Genre	Publisher	Rating
Grand Theft Auto: San Andreas	Action	Take-Two Interactive	AO
Supreme Ruler: Cold War	Strategy	Paradox Interactive	RP

Grand Theft Auto: San Andreas was famously revised from M to an AO rating temporarily after a mod for the game was release that unlocked explicit material unused in the unmodified game. After some back and forth this content was eventually removed from distributed copies and the original M rating was restored. We will make the same modification hear to better model the true outcome.

Digging further into other Strategy titles published by Paradox Interactive similar to *Supreme Ruler: Cold War*:

Name	Genre	Rating
Hearts of Iron III	Strategy	E10+
Hearts of Iron IV	Strategy	NA
Stellaris	Strategy	NA
Supreme Ruler: Cold War	Strategy	RP
Combat Mission: Shock Force	Strategy	T
King Arthur II	Strategy	NA
Commander: Conquest of the Americas	Strategy	T
Europa Universalis III Complete	Strategy	E10+
Elven Legacy	Strategy	T
East India Company	Strategy	T
Elven Legacy Collection	Strategy	NA
Majesty 2 Collection	Strategy	T
Sengoku	Strategy	T

We see mostly titles rated T for Teen with a handful making it under the T for Teen bar after the E10+ rating is introduced. We will use T for this RP title to approximate the rating it would have received.

Finally we will need a rating to assign the many unrated titles there are thousands and we have little to go on so for the sake of expediency we will lump them together as Unrated designated as "UR." The ESRB is primarily a western market organization so the expectation is these titles will effectively be a group for everything not released in those markets.

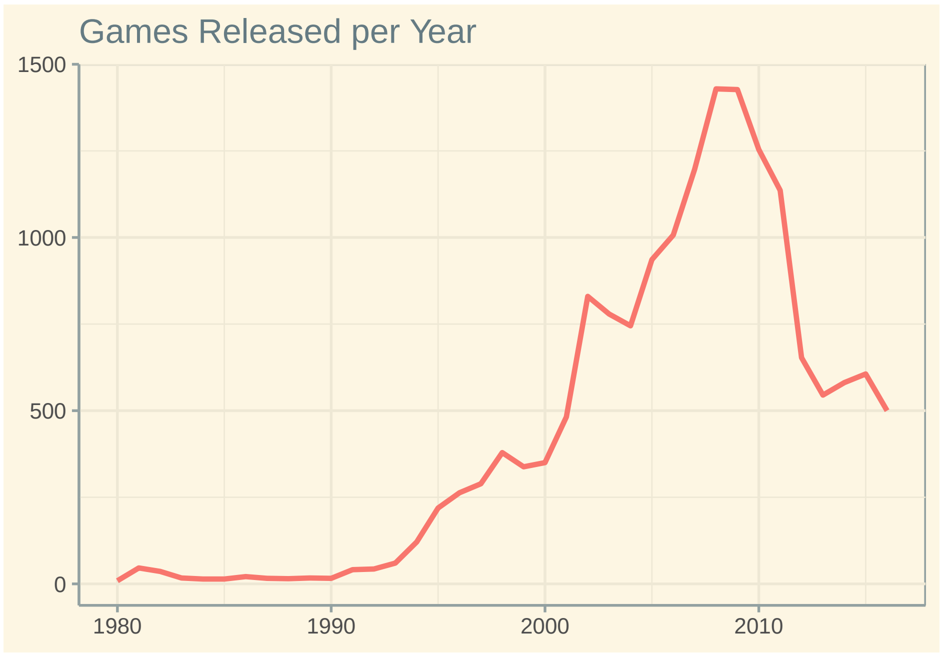
	#_of_NAs
Critic_Score	8436
Critic_Count	8436
User_Score	8958
User_Count	8958

A final check of our data shows we have cleaned up all the NAs from fields we intend to use in our models. Lets take a deeper look at the shape of our data next.

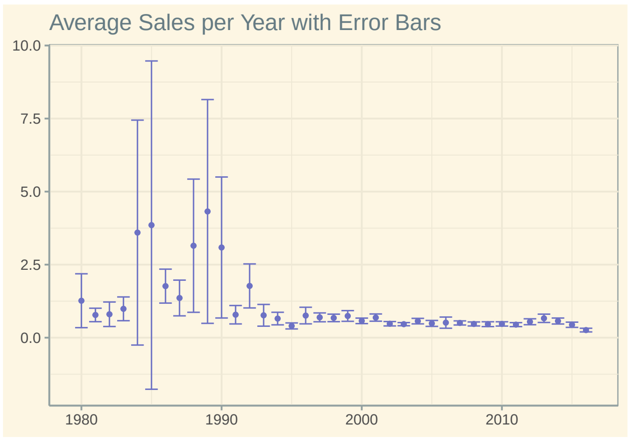
Data Visualization

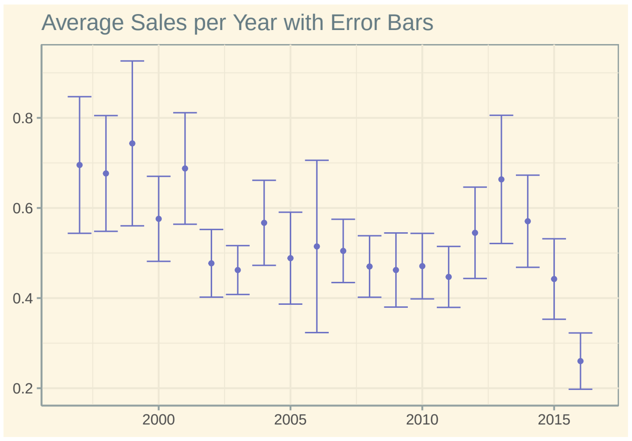
Let us explore our data set with some graphical representations to get a better idea of the data's shape and which fields may be useful for our work. We will examine each potential predictive factor as well as the global sales figures we are trying to predict to see if any insights can be gleaned.

Sales by Year of Release Starting with the number of titles released per year in our data set which we can see ploted over time here:



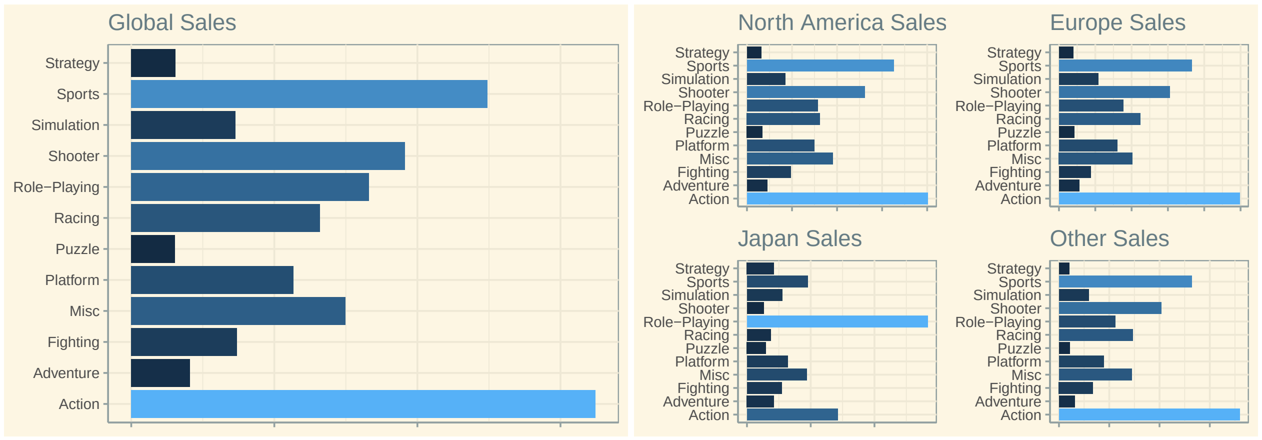
We can see that very little data exists in our data set earlier than 1993 and for 1993-2000 or so the data in the set is relatively low compared to later years. How much does this impact the quality of our data? We will graph the average sales by year below on the left with error bars to determine how the data quality is affected by the small sample sizes. As expected the early years are spotty and have wide variances in the data. Prior to 1997 the error bars ar e quite large compared to later years which are extremely compressed on this graph due to the scale required for the early years. The since the further back the data goes the less likely it is a good predictor for future forecasts and the quality of the data in this period is low we will discard the data from 1997 or earlier.





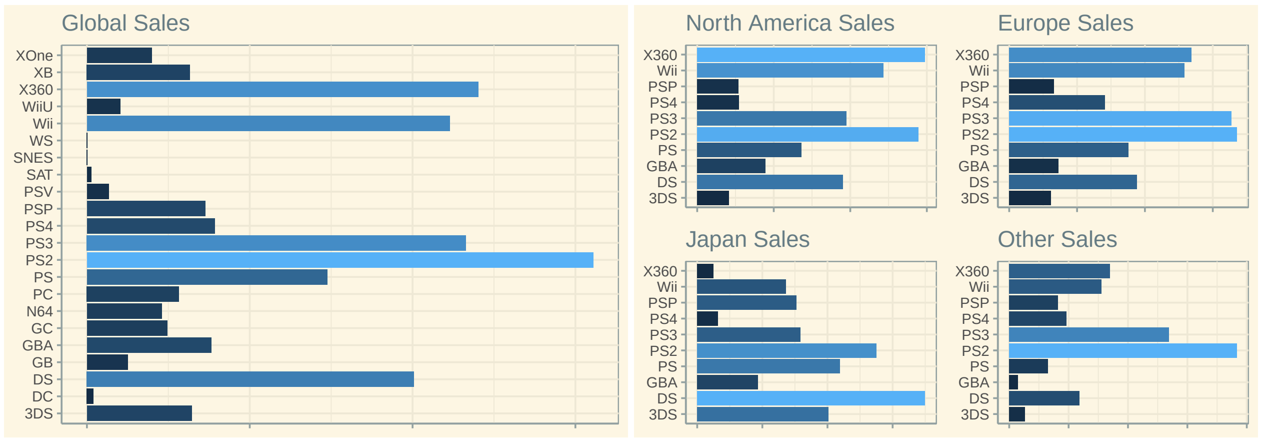
Looking at the results after cleaning on the right, we can see our y-axis has tightened up considerably and all years are showing similar sizes of variance. This should make for a more accurate model.

Genre The first variable we will examine is Genre. Here we will chart the total sales by genre first globally and then a chart broken down by region.



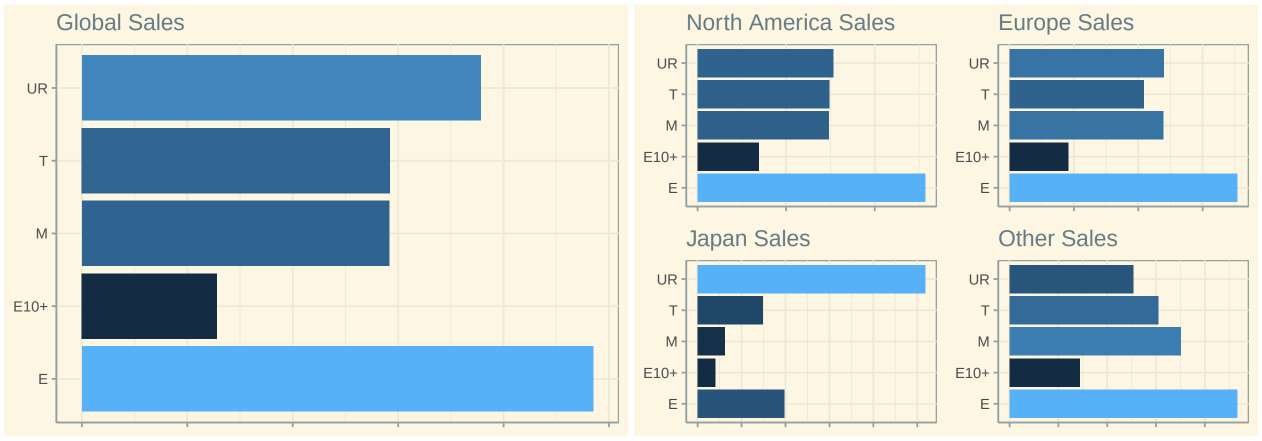
We can see some patterns here in what regions of the world tend to purchase which genres of games. Japan heavily favors Role-Playing games while the rest of the world prefers Action games. Sports and Shooters are also very popular outside of Japan. The European region also has a slightly above average preference for Racing games.

Platform Next we will examine is Genre. Again we will chart the total sales by platform first globally and then a chart broken down by region.



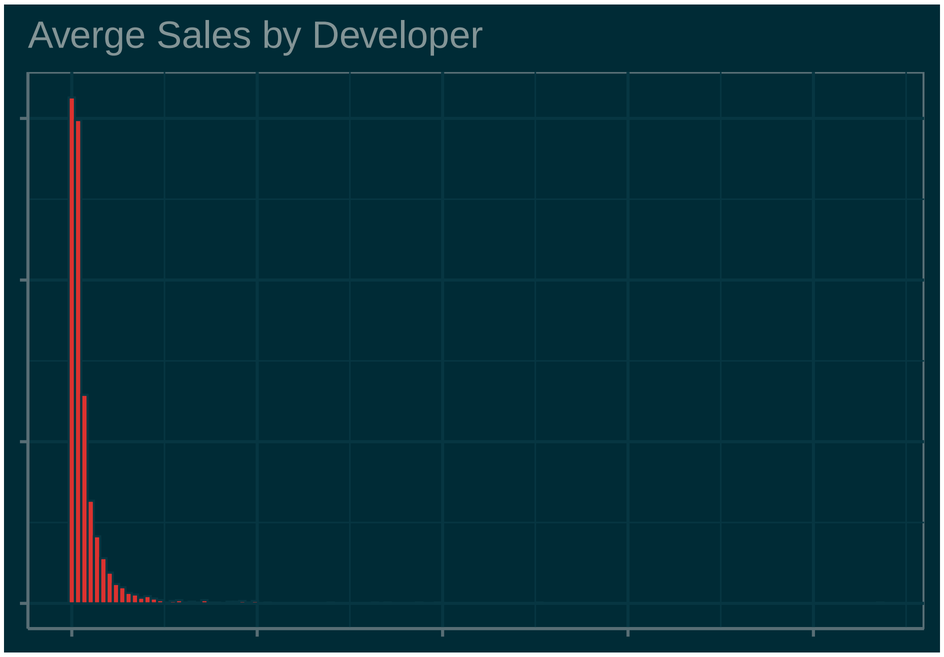
Again we see distinct regional patterns in Platform adoption. Japan heavily prefers the DS. Outside Japan the PS2 enjoys wide adoption everywhere, the Wii is popular in US and the EU but not as popular outside those regions, and the Xbox 360 is most popular in the US with the PS2 and PS3 edging above it in Europe. Japan also appears to be less enthusiastic about the latest Play Station generation with PS3 titles still lagging behind original Play Station titles.

Rating Next we examine rating again breaking down the results globally and then by region.



Regional patterns in preference for specific age ratings for games appears to be laregely non-existant outside of Japan with almost every region seeing a distinct pattern of E, M and T as the top 3. Keep in mind when reviewing these numbers that E10+ is still relatively new so the numbers have not caught up with other ratings. Inside Japan we see the mystery of our large numbers of unrated games revealed: most of Japan's games sales in this data set lack ESRB ratings. This makes sense because the ESRB primarily operates in the US and Canada. In the EU they work closely with PEGI which uses a nearly iddentical rating system of 3, 7, 12, 16, 18 maping approximately onto E, E10+, T, M, and AO. So the data set has ratings for those titles as well.

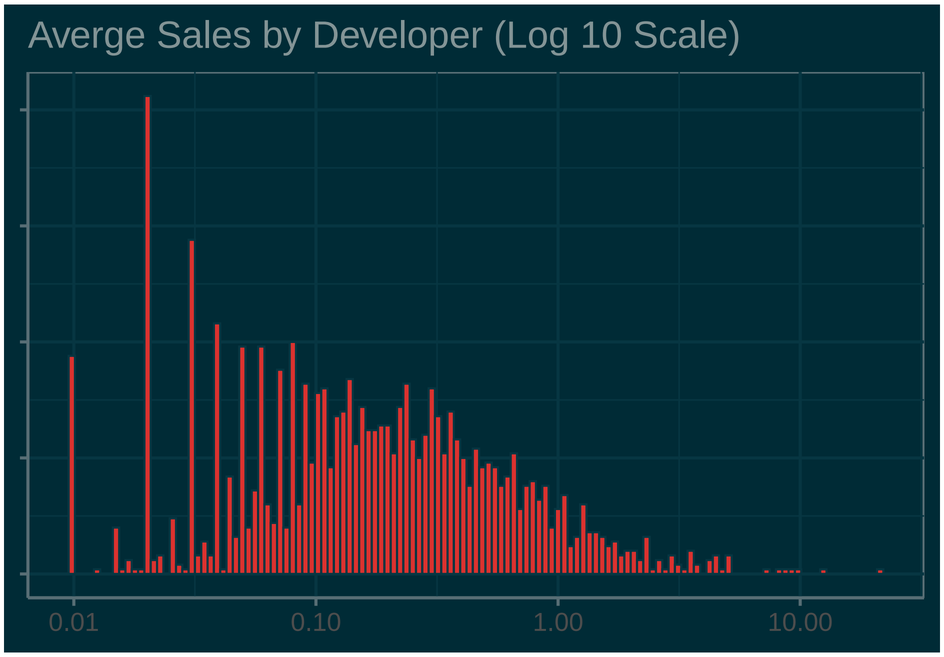
Developer Next we examine how possible Developer may affect sales. This could be a name brand or reputation based effect. We can see a heavy leftward bias in our chart with some extreme outliers pulling the x-axis scale very far to the right with very few data points.



Developer	average_sales
Good Science Studio	21.810000
Retro Studios, Entertainment Analysis & Development Division	12.660000

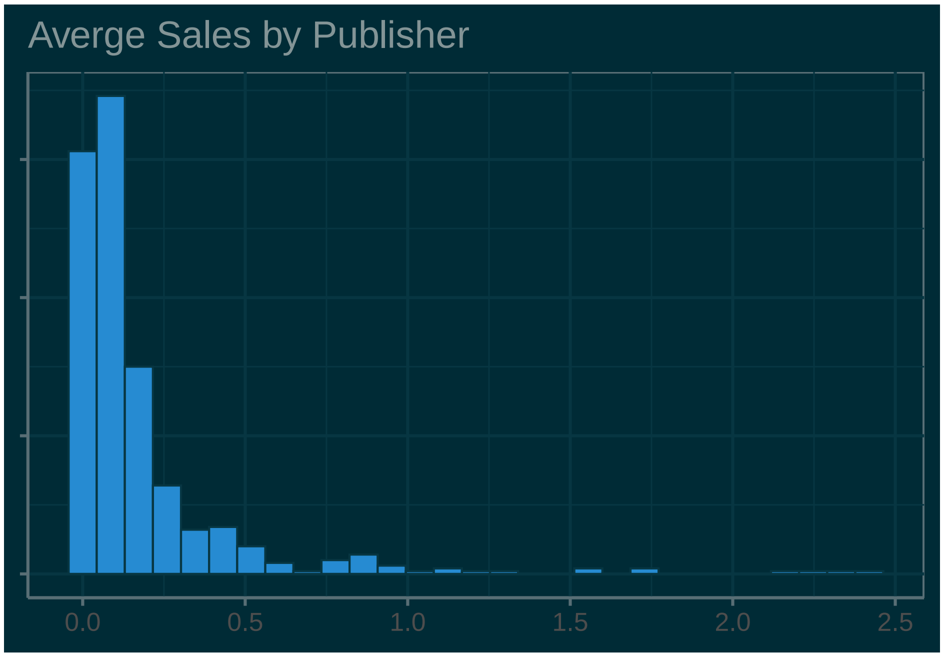
Developer	average_sales
Infinity Ward, Sledgehammer Games	9.923333
Polyphony Digital	9.314286
Rockstar North	8.533571
DMA Design	8.260000
Game Freak	7.065000
Bungie Software	5.203333
Neversoft Entertainment, BudCat	4.980000
343 Industries	4.955000

Checking the top 10 Developers by average sales we can see some clear outliers at the top of the list here with very popular titles getting tens of millions of units sold while the average game according to our chart sells less than a million units. This suggests a scale adjustment is in order, let's try log10 and see how things look.



After a log10 scale adjustment we can see our distribution now looks a little closer to roughly normal by still skews a bit left. This should make guessing an average much easier.

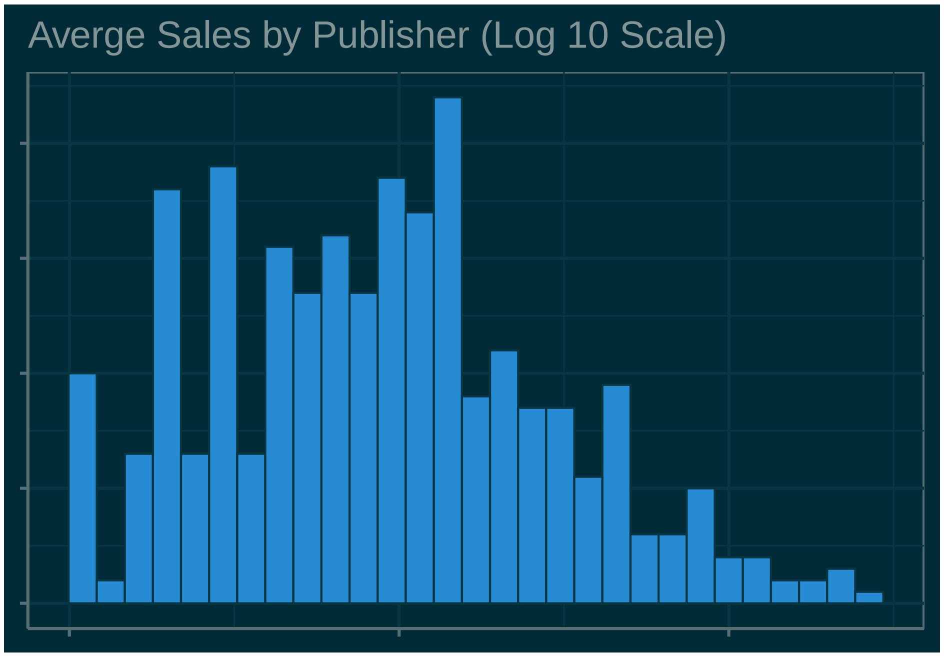
Publisher Next we examine how possible Publisher may affect sales. This could be a name brand or reputation based effect. Publishers are also typically responsible for the marketing of a game so this could also reflect the differences in marketing styles between the Publishers.



Again we see a heavy leftward bias in our sales figures with some extreme outliers pulling the x-axis scale very far to the right. Checking the top 10 Publishers by average sales per game:

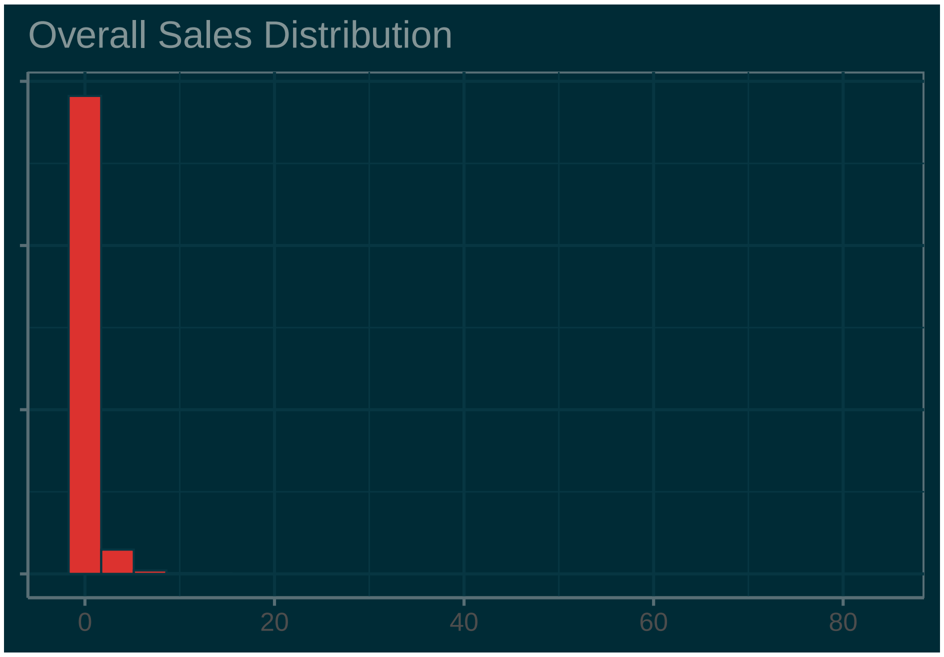
Publisher	average_sales
Red Orb	2.430000
Nintendo	2.291503
UEP Systems	2.250000
RedOctane	2.170000
Hello Games	1.700000
Valve	1.700000
Sony Computer Entertainment Europe	1.558000
Westwood Studios	1.550000
Microsoft Game Studios	1.280000
Black Label Games	1.200000

Shows a similar story to Developers with a couple of outliers at the top averaging several million units sold per game. Lets try a log10 adjustment as we did before.

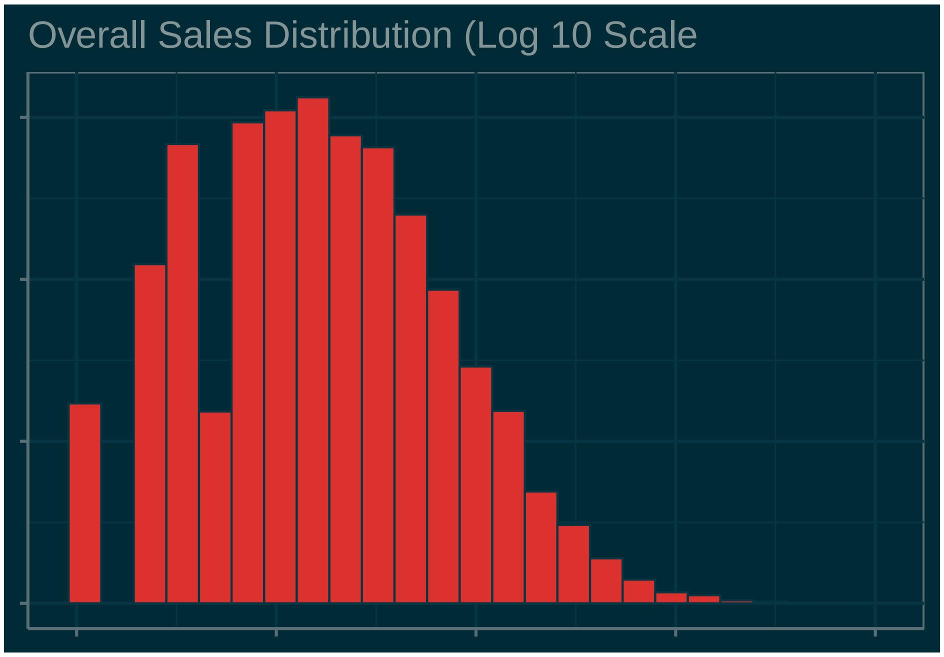


After a log10 scale adjustment we can see our distribution now looks a little closer to roughly normal but still skews a bit left. Still an improvement over the unsealed results and should be easier to guess a mean for.

Global Sales Distribution Lets examine the distribution of Global Sales directly.



Again we see a pretty tight grouping near 0 with very few data points skewing the x axis to the right considerably. Applying a log10 scale:



Much closer to a normal distribution now. Still a left skew but much less pronounced. This should make the data easier to model with a mean based method. We will take this conclusion in account and log adjust our Global Sales before building our models.

Models

After examining our data we have identified Platform, Year_of_Release, Genre, Publisher, Developer, Rating as potential predictors and that our sale figures need a logarithmic adjustment before building our models. We will apply that adjustment and eliminate our unused columns then cut our data set into 2 groups, a 90% train set and a 10% validation set. The validation set will only be used to check the final results of the model and not for training or cross validation. For cross validation we will further divide our train set into an 80% train set and a 20% test set. This will allow us to check how well our model is performing during the creation process as well as tune our lambda for regularization of the linear model and the mtry, the number of variables randomly sampled as candidates at each tree split, of our random forest model. During our analysis we will check our progress using a Root Mean Squared Error function.

```r
# identified possible predictors: Platform, Year_of_Release, Genre, Publisher, Developer, Rating
# identified sales needed a log10 adjustment
vgs <- vgs %>% mutate(Global_Sales = log10(Global_Sales)) %>% select(2,3,4,5,15,16,10)

# data for revenue prediction model
# sales_validation set will be 10% of vgs data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = vgs$Global_Sales, times = 1, p = 0.1, list = FALSE)
sales_train <- vgs[-test_index,]
temp <- vgs[test_index,]

# Make sure all predictors in sales_validation set are also in sales_train set
sales_validation <- temp %>%
  semi_join(sales_train, by = "Platform") %>%
  semi_join(sales_train, by = "Year_of_Release") %>%
  semi_join(sales_train, by = "Genre") %>%
  semi_join(sales_train, by = "Publisher") %>%
  semi_join(sales_train, by = "Developer") %>%
  semi_join(sales_train, by = "Rating")

# Add rows removed from sales_validation set back into sales_train set
removed <- anti_join(temp, sales_validation, by = c("Platform", "Year_of_Release", "Genre", "Publisher"
sales_train <- rbind(sales_train, removed)

# divide sales_train into cross validation and train tests
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = sales_train$Global_Sales, times = 1, p = 0.2, list = FALSE)
sales_train_cv <- sales_train[-test_index,]
temp <- sales_train[test_index,]

# Make sure all predictors in sales_test_cv set are also in sales_train_cv set
sales_test_cv <- temp %>%
  semi_join(sales_train_cv, by = "Platform") %>%
  semi_join(sales_train_cv, by = "Year_of_Release") %>%
  semi_join(sales_train_cv, by = "Genre") %>%
  semi_join(sales_train_cv, by = "Publisher") %>%
  semi_join(sales_train_cv, by = "Developer") %>%
  semi_join(sales_train_cv, by = "Rating")

# Add rows removed from sales_test_cv set back into sales_train_cv set
removed <- anti_join(temp, sales_test_cv, by = c("Platform", "Year_of_Release", "Genre", "Publisher", "
sales_train_cv <- rbind(sales_train_cv, removed)

# dump extra variables
rm(test_index,temp,removed)
```

```
# create RMSE function to evaluate model results
rmse <- function(x, y)
  {
  sqrt(mean((x - y)^2))
  }
```

Model 1: Linear Regression with Regularization

We begin by guessing a simple mean of the Global Sales figures for each entry.

Method	RMSE
Just the average	0.6351377

This gives us a baseline RMSE of 0.6351377 to compare our models against.

Developer First we will build a model using Developer as a predictor and check RMSE.

Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952

We immediately see an improvement to 0.5412952.

Publisher Next we add Publisher on top of the existing model.

Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952
Dev + Pub Model	0.5376022

Another improvement to 0.5376022 but a much smaller jump this time.

Genre Adding Genre as a variable on top of the existing model.

Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952
Dev + Pub Model	0.5376022
Dev + Pub + Genre Model	0.5363075

Another improvement to 0.5363075

Platform Next we add Platform as a variable on top of the existing model.

Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952
Dev + Pub Model	0.5376022
Dev + Pub + Genre Model	0.5363075
Dev + Pub + Genre + Platform Model	0.5184477

Another improvement to 0.5184477

Rating Next we add Rating to our model.

Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952
Dev + Pub Model	0.5376022
Dev + Pub + Genre Model	0.5363075
Dev + Pub + Genre + Platform Model	0.5184477

Method	RMSE
Dev + Pub + Genre + Platform + Rating Model	0.5160104

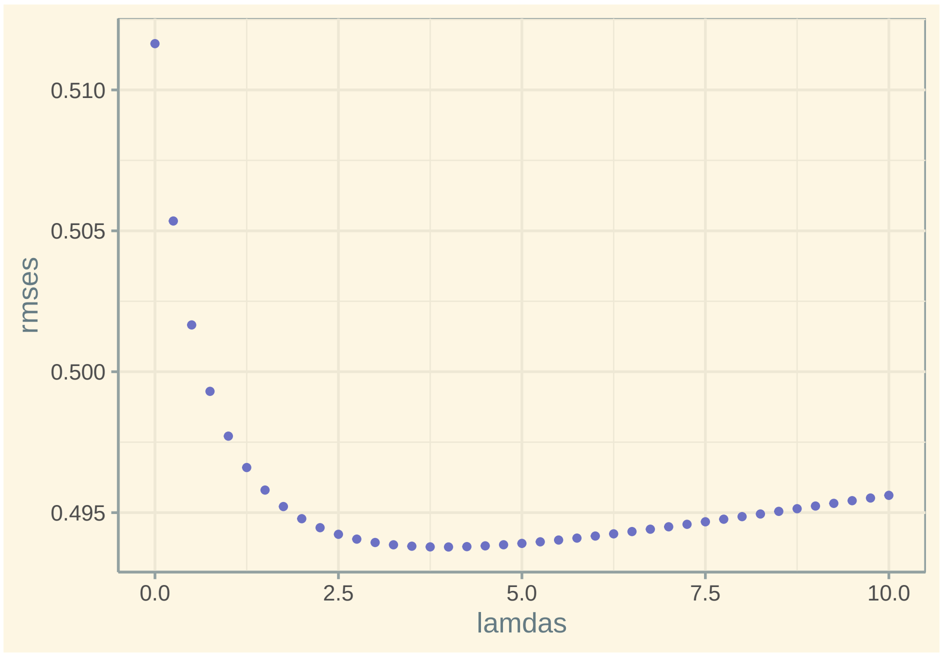
Another improvement to 0.5160104

Year of Release Our final predictor, the Year of Release, will be added to the model.

Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952
Dev + Pub Model	0.5376022
Dev + Pub + Genre Model	0.5363075
Dev + Pub + Genre + Platform Model	0.5184477
Dev + Pub + Genre + Platform + Rating Model	0.5160104
Dev + Pub + Genre + Platform + Rating + Year Model	0.5116427

Yielding a final RMSE of 0.5116427 with each predictor improving the model.

Regularization Next we will apply regularization, a technique that prevents over-fitting and pushes the model back towards the mean again. We will use our cross validation set to tune our lambda, the factor we will add to each group to regularize them.



Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952
Dev + Pub Model	0.5376022

Method	RMSE
Dev + Pub + Genre Model	0.5363075
Dev + Pub + Genre + Platform Model	0.5184477
Dev + Pub + Genre + Platform + Rating Model	0.5160104
Dev + Pub + Genre + Platform + Rating + Year Model	0.5116427
Regularized Dev + Pub + Genre + Platform + Year Model	0.4937804

Here we can see the results of our cross validation pass where different levels of lambda result in different RMSE results. We pick the best lambda, 4, from the bunch, store the RMSE to our table, and compare it to the earlier iterations of the model and see another improvement to 0.4937804.

Model 2: Random Forests

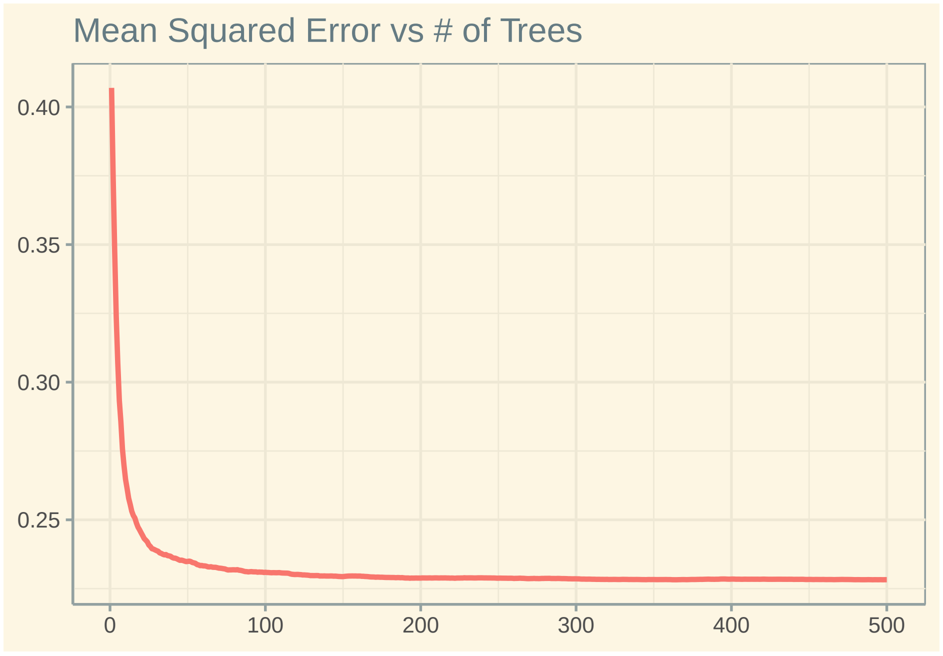
Our next model will use a technique called Random Forests. Random forests creates a large number of decision trees to model the prediction data and then selects the average result from all the predictions produced to arrive at a final prediction for each entry. Again we will measure the effectiveness of this model using the same Root Mean Squared Error function as before so the two models can be compared head to head.

Convert Categories with a Hash Before we can begin building our Random Forest we need to convert some of our text based categorical data into a hash the algorithm can handle. For our Developer and Publisher fields there are two many categories for the algorithm to interpret as categorical data so this will allow us to still use them as predictors.

```r
# lets try another model, random forests
# start with hashing of developer and publisher names
# because there are to many categories for RF
sales_train_cv <- sales_train_cv %>%
  rowwise() %>%
  mutate(digest_dev = digest(Developer, algo = 'md5'),
         digest_pub = digest(Publisher,algo='md5'),
         digest2int_dev = digest2int(digest_dev),
         digest2int_pub = digest2int(digest_pub)) %>%
  as.data.frame(.)
sales_test_cv <- sales_test_cv %>%
  rowwise() %>%
  mutate(digest_dev = digest(Developer, algo = 'md5'),
         digest_pub = digest(Publisher,algo='md5'),
         digest2int_dev = digest2int(digest_dev),
         digest2int_pub = digest2int(digest_pub)) %>%
  as.data.frame(.)
sales_validation <- sales_validation %>%
  rowwise() %>%
  mutate(digest_dev = digest(Developer, algo = 'md5'),
         digest_pub = digest(Publisher,algo='md5'),
         digest2int_dev = digest2int(digest_dev),
         digest2int_pub = digest2int(digest_pub)) %>%
  as.data.frame(.)
# set y
rf_y <- sales_train_cv$Global_Sales
# linear regression suggests Developer, Publisher, Genre,
# Platform, Rating, and Year_of_Release as variables
rf_x <- sales_train_cv %>%
  select(digest2int_dev,digest2int_pub,Genre,Platform,Rating,Year_of_Release)
```

Training Now we know from our linear model that all of the selected predictors have some predictive power so for random forests we will skip to the end and include them all at the start and begin building our model. At the early stages we will limit the sample size the model uses when drawing. This will speed up processing. After an initial run we will examine the effect of the number of trees on the error.

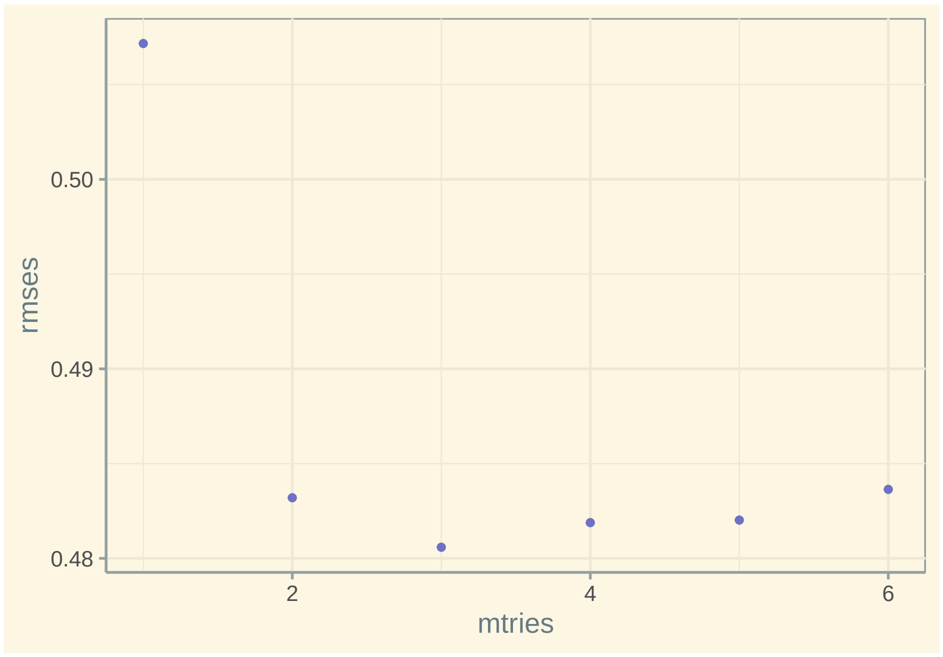
```r
# set seeed for repeatability
set.seed(1, sample.kind="Rounding")
# train the random forest, limit sample size to improve performance
train_rf <- randomForest(rf_x,rf_y,sampsize=6500)
# calculate RMSE
rf_rmse <- rmse(predict(train_rf, sales_test_cv),sales_test_cv$Global_Sales)
# plot train set to examine number of trees for cv
as.data.frame(train_rf$mse) %>% mutate(Error = train_rf$mse, num_trees=row_number()) %>%
  ggplot(aes(x=num_trees,y=Error,color = "num_trees")) +
  theme_solarized(base_size = 14) +
  ggtitle("Mean Squared Error vs # of Trees") +
  geom_line(show.legend=FALSE, size = 1.25) +
  theme(axis.title = element_blank())
```



We can see from the chart that the error function sharply drops and stabilizes at around 100-150 trees. We can use this as a limit to help speed up the cross validation pass on our tuning parameter, mtry. Mtry is the number of variables randomly sampled each time the data is split for branches of our decision trees. The default value, used here, is the number of predictors divided by 3. We have 6 predictors so our mtry is 2. We will see if we can do better by selecting mtry manually using cross validation similar to the process we used to pick lambda earlier.

```r
# use cross validation to choose mtry
mtries <- 1:6
rmses <- sapply(mtries, function(n)
{
  set.seed(1, sample.kind="Rounding")
  cv_tune <- randomForest(rf_x,rf_y,
                          sampsize=6500,
                          ntree=100,
                          mtry = n)
  return(rmse(predict(cv_tune, sales_test_cv),sales_test_cv$Global_Sales))
})
# plot to visualize effect of mtry on RMSE
as.data.frame(list(mtries = mtries,rmses = rmses)) %>%
  ggplot(aes(x=mtries,y=rmses)) +
  geom_point(show.legend=FALSE,color = "#6c71c4") +
  theme_solarized(base_size = 14)
```

Tune Mtry



```r
# store best mtry
mt <- mtries[which.min(rmses)]
```

Here we test all possible values of mtry, 1-6, against the data with the number of trees limited to 150 to help speed up processing. We calculate the RMSE of each value of mtry and then pick the mtry value with the best result. After cross validation our best mtry value was 3.

Final Random Forest Model Finally it is time to take our model and tuned variable and build the best random forest model we can. We lift the sample size limits and raise the number of trees to 2,500 here to improve model. As a result this calculation will take several minutes.

```r
# lift sample size and raise ntree limits for final tune
# this will take several minutes
set.seed(1, sample.kind="Rounding")
train_rf_tuned <- randomForest(rf_x,rf_y,
                               mtry = mt,
                               ntree=2500)
# calculate RMSE
rf_rmse_tuned <- rmse(predict(train_rf_tuned, sales_test_cv),sales_test_cv$Global_Sales)
# store rmse from final rf model to table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Tuned Random forest Model",
                                 RMSE = min(rmses)))
# full rmse table from model building
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	0.6351377
Dev Model	0.5412952
Dev + Pub Model	0.5376022
Dev + Pub + Genre Model	0.5363075
Dev + Pub + Genre + Platform Model	0.5184477
Dev + Pub + Genre + Platform + Rating Model	0.5160104
Dev + Pub + Genre + Platform + Rating + Year Model	0.5116427
Regularized Dev + Pub + Genre + Platform + Year Model	0.4937804
Tuned Random forest Model	0.4805872

Our final RMSE from Random Forset gives a RMSE against the cross validation test set of 0.4805872. A small improvement over our best linear model with regularization.

Results

With both models build and tuned we will take them back to the hold-out validation set for the first time to guage their performance

Linear Regression with Regularization Validation and Final RMSE

First we calculate the RMSE of guessing the average of the training set against the validation set as a baseline to compare against. Then we calculate our Linear Regression with Regularization model against the full training set, use the model to predict the Validation set, and calculate a RMSE of our predictions.

```r
# with our CV complete, model and lambda set, apply model to full train set
# mean of full train set
mu <- mean(sales_train$Global_Sales)
# first a naive guess of the simple mean for comparison
final_naive_rmse <- rmse(sales_validation$Global_Sales, sales_mu )
# add result to table
final_rmse_results <- tibble(Method = "Just the average", RMSE = final_naive_rmse)
# create full model on full train set with lambda
b_dev <- sales_train %>%
  group_by(Developer) %>%
  summarize(b_dev = sum(Global_Sales - mu)/(n()+lambda))
b_pub <- sales_train %>%
  left_join(b_dev, by="Developer") %>%
  group_by(Publisher) %>%
  summarize(b_pub = sum(Global_Sales - b_dev - mu)/(n()+lambda))
b_g <- sales_train %>%
  left_join(b_dev, by="Developer") %>%
  left_join(b_pub, by="Publisher") %>%
  group_by(Genre) %>%
  summarize(b_g = sum(Global_Sales - b_dev - b_pub - mu)/(n()+lambda))
b_plat <- sales_train %>%
  left_join(b_dev, by="Developer") %>%
  left_join(b_pub, by="Publisher") %>%
  left_join(b_g, by="Genre") %>%
  group_by(Platform) %>%
  summarize(b_plat = sum(Global_Sales - b_dev - b_pub - b_g - mu)/(n()+lambda))
b_rt <- sales_train %>%
  left_join(b_dev, by="Developer") %>%
  left_join(b_pub, by="Publisher") %>%
  left_join(b_g, by="Genre") %>%
  left_join(b_plat,by="Platform") %>%
  group_by(Rating) %>%
  summarize(b_rt = sum(Global_Sales - b_dev - b_pub - b_g - b_plat - mu)/(n()+lambda))
b_y <- sales_train %>%
  left_join(b_dev, by = "Developer") %>%
  left_join(b_pub, by = "Publisher") %>%
  left_join(b_g, by = "Genre") %>%
  left_join(b_plat, by = "Platform") %>%
  left_join(b_rt, by = "Rating") %>%
  group_by(Year_of_Release) %>%
  summarize(b_y = sum(Global_Sales - b_dev - b_pub - b_g - b_plat - b_rt - mu)/(n()+lambda))
predicted_global_sales <- sales_validation %>%
  left_join(b_dev, by = "Developer") %>%
  left_join(b_pub, by = "Publisher") %>%
  left_join(b_g, by = "Genre") %>%
  left_join(b_plat, by = "Platform") %>%
  left_join(b_rt, by = "Rating") %>%
  left_join(b_y, by = "Year_of_Release") %>%
  mutate(pred = mu + b_dev + b_pub + b_g + b_plat + b_rt + b_y) %>%
  .$pred
# calculate final RMSE against validation set
final_linear_rmse <- RMSE(predicted_global_sales, sales_validation$Global_Sales)
# save final rmse to table
final_rmse_results <- bind_rows(final_rmse_results,
                          tibble(Method="Linear Regression with Regularization Model",
                                 RMSE = final_linear_rmse ))
# final rmse results table
final_rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	0.6473588

Method	RMSE
Linear Regression with Regularization Model	0.4803899

We get a result of "r final_linear_rmse" against the validation set, an improvement over the naive estimate.

Random Forests Validation and Final RMSE

```r
## calculate final RMSE against validation set
final_rf_rmse <- rmse(predict(train_rf_tuned, sales_validation),sales_validation$Global_Sales)
# save final rmse to table 0.4786551
final_rmse_results <- bind_rows(final_rmse_results,
                                tibble(Method="RF with all varaibles",
                                       RMSE = final_rf_rmse ))
# final table
final_rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	0.6473588
Linear Regression with Regularization Model	0.4803899
RF with all varaibles	0.4786551

Next we use our Random Forests model to guess the validation set and calculate an RMSE arriving at 0.4786551 and improvement over both the naive estimate as well as the previous model.

Conclusion

We took a data set of video game sales data and used six attributes from each title to create two regression models, one linear and one random forest, that attempted to predict Global Sales. We used a root mean squared error (RMSE) loss function to evaluate these models against one another to compare how effective they were at predicting sales. In our analysis Random Forests resulted in a slightly more accurate estimate but with a significant amount of the variance still left unexplained. Still a significant improvement was made over a naive average estimate.

Limitations

Many attributes of games are not captured by the variables in our model and indeed could not be known before development even begins such as reception to the visual presentation or game-play. Reviews could help with this analysis and indeed our data set does contain review scores but such scores could only be set after the development work is largely complete. The data set also stops at 2016 so the conclusions on the present day are somewhat limited. Lastly the entire premise of our regression models is to predict future sales with existing information so titles from new developers, new publishers, for new platforms, or pushing into new genres would be difficult for these models to accurately predict.

Future Work

Cross referencing this data set with other sources of information the various attributes of these video game titles could yield interesting results. Information like the engine used, more detailed information on who developed the game beyond just a studio name down to a more granular level such as Head Writers, Directors, or Art Designers, or who voiced the characters in the game could also yield valuable predictive insights.

References

1. Rafael A. Irizarry (2022) Introduction to Data Science Data Analysis and Prediction Algorithms with R
2. Rush Kirubi (2016) Video Game Sales with Ratings
3. U.S. Bureau of Labor Statistics (2022) U.S. city average – Current – All items less food and energy – Monthly – Not Seasonally Adjusted CUUR0000SA0L1E
4. ESRB (2022) Entertainment Software Rating Board - Our History
5. PEGI (2022) https://pegi.info/what-do-the-labels-mean
6. RDocumentation (2022) randomForest: Classification and Regression with Random Forest
7. RDocumentation (2022) digest: Create hash function digests for arbitrary R objects or files