

Game_Sales_Report

Group 18: Adeer Siddiqui, Tuan Pham, Uyen Vi Phan
Spring 2025

1. Introduction // Tuan Pham

For our project, we decided to analyze the `global video game sales` data from 2024, sourced from Kaggle.com. Our inspiration comes from our love for video games and we are curious about which factors determine a video game's success. This dataset has `64017 rows` with `14 columns`. The variables are listed below:

`img`: An url to an image of the game packaging on vgchartz.com

`title` : The title of the game

`console` : The console the game is released for

`genre` : The genre of the game

`publisher` : The publisher of the game

`developer`: The developer of the game

`critic_score`: The metacritic score (out of 10)

`total_sales` : Total sales globally in millions of USD

`na_sales` : North American sales in millions of USD

`jp_sales` : Japanese sales in millions of USD

`pal_sales` : European & African sales in millions of USD

`other_sales` : Rest of world sales of copies in millions of USD

`release_date` : Date the game was released on

`last_update` : Date the data was last updated

Of the 14 variables, our group will most likely be examining the `console`, `genre`, `publisher`, `developer`, `critic score`, and `release date` (mostly focusing on the month) as potential predictors. The `title` of the game and `img` serve as an identifier and are solely unique to each game, so they will not contribute to sales prediction. Sales in individual countries are irrelevant as we are focused on predicting `total_sales`. We drop the `last_update` variable because it likely reflects when the dataset entry was edited, not something about the game itself. It's not informative for sales analysis and keeping it might mislead people into thinking it's an in-game update date, which it's probably not. Our response variable will be `total_sales`.

Our primary objective is to predict video game sales based on `genre`, `publisher`, `developer`, `critic score`, and `release month`. Our main question is: what is the predicted total sales of a game based on `genre`, `publisher`, `developer`, `critic score`, and `release month`. The models we want to examine for this project are Linear Regression and Decision Tree, and use cross-validation for model comparison. Using both models together allows us to compare results and identify the best approach for predicting game sales.

There are 4 main tasks for this project: data preparation, creating the first model, creating the second model, and interpreting the models into the report. Tuan Pham worked on data preparation and assist the other group members build their models. Uyen Vi Phan created and analyzed the Linear Regression model, and Adeer Siddiqui created and analyzed the Decision Tree Model. Then our group combined and discussed our works to make the final report.

2. Linear Regression // Uyen Vi Phan

Linear Regression is a parametric model that seeks to represent the relationship between the output and input variables linearly. Our objective is to be able to predict and find the factors that affect the `total_sales` of [global video game sales data from 2024](#). Our group decided to choose Linear Regression to model our data because it provides a simple linear equation that can be easily calculated by hand. It gives us weights for each predictor that indicate how much `total_sales` changes with one unit increase or decrease of each of our predictors.

In addition to this, Linear Regression offers a simple and interpretative method that can be used to predict `total_sales`. However, due to its simplicity, it has trouble effectively modeling complex data and the data must fit specific assumptions for Linear Regression to be effective: the predictors must have a linear relationship with `total_sales`, each observation must be independent of one another, the data must be normally distributed, and they must have equal variance.

a. Model Equation

The Linear Regression model equation is displayed below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

Where y represents our output variable, `total_sales`, β_0 represents our model intercept, β_i is our coefficient for our predictor x_i and ϵ is our error term.

b. Process

In order to properly fit the model to our data, we must exclude some of our data that may have no affect on the `total_sales`. Our initial data cleanup already removed a lot of unnecessary variables, but we could go even further. I initially attempted to pick which predictors to keep using their p-values. I quickly realized that since our data has so many categorical variables with a lot of categories, eliminating predictors based on p-values was not efficient. The output is withheld due to the length of the results.

```
library(readr)
game <- read_csv("game.csv")
attach(game)

game.lm = lm(total_sales ~ ., data = game)
summary(game.lm)
```

The method for determining what predictors are significant in predicting `total_sales` that worked is backwards stepwise regression using the `step()` function, which ended up working fine. Stepwise regression concluded that `console`, `genre`, `publisher`, `critic_score` and `release_month` were significant in predicting `total_sales`. The output is withheld due to the length of the results.

```
step(game.lm)
```

However, when it came to validating the data, another problem came up. Since our data has so many categorical variables with many different categories, sometimes all the observations under one category could end up in the test set and never appear in the training set, leading to an error because the model is not trained to predict `total_sales` using that specific category. To remedy this, I first looked at all the categories from each categorical predictor and checked their frequencies. I decided remove `publisher` and `developer` from consideration, as they had over 200 different categories. Including them could lead to overfitting our model. `genre` and `console` only had less than 30 categories, so they could still be significant in predicting `total_sales`. I then repeated the `step()` function again on the remaining variables. The `step()` function concluded that `console`, `genre`, `critic_score` and `release_month` were significant in predicting `total_sales`.

```
game.pub.types = table(game$publisher)
View(game.pub.types)
game.dev.types = table(game$developer)
View(game.dev.types)
game.gen.types = table(game$genre)
View(game.gen.types)
game.con.types = table(game$console)
View(game.con.types)

game.lm = lm(total_sales ~ console+genre+critic_score+release_month, data = game)
step(game.lm)
```

Start: AIC=1901.92

`total_sales ~ console + genre + critic_score + release_month`

	Df	Sum of Sq	RSS	AIC
<none>			6386.0	1901.9
- genre	19	142.86	6528.9	1955.1
- release_month	1	141.45	6527.5	1990.2
- critic_score	1	556.54	6942.5	2244.4
- console	27	769.32	7155.3	2316.9

Call:

```
lm(formula = total_sales ~ console + genre + critic_score + release_month,  
    data = game)
```

Coefficients:

(Intercept)	consoleDC	consoleDS
-1.67014	-0.38617	0.13866
consoleGB	consoleGBA	consoleGBC
0.38812	-0.13839	0.98775
consoleGC	consoleGEN	consoleN64
-0.22756	-0.72350	-0.05821
consoleNES	consoleNS	consolePC
4.11348	-0.24612	-0.35629
consolePS	consolePS2	consolePS3
0.34353	0.60969	0.61781
consolePS4	consolePSN	consolePSP
1.96016	0.83943	0.07117
consolePSV	consoleSAT	consoleSNES
0.22717	-0.49488	-0.60963
consoleVC	consoleWii	consoleWiiU
-0.63603	0.23719	-0.22026
consoleX360	consoleXB	consoleXBL
0.46480	-0.17986	-0.58745
consoleXOne	genreAction-Adventure	genreAdventure
0.73444	-0.32226	-0.22196
genreBoard Game	genreEducation	genreFighting
0.22352	-0.67928	-0.28220
genreMisc	genreMMO	genreMusic
-0.13909	-0.94829	-0.46788
genreParty	genrePlatform	genrePuzzle
0.08776	-0.19890	-0.40943
genreRacing	genreRole-Playing	genreSandbox
-0.20181	-0.37153	0.99512
genreShooter	genreSimulation	genreSports
0.22915	-0.03644	-0.12828
genreStrategy	genreVisual Novel	critic_score
-0.40201	-1.28551	0.26836
release_month		
0.05705		

c. Validation

In order to verify how well our model predicts `total_sales` to new data, we must perform cross-validation to obtain the average Mean Squared Error. For this model, we performed the validation method 10 times. Despite removing `publisher` and `developer` from the model, we still had the issue where whole categories will go into the testing set, however not to as a severe degree as before. Because of this, it was easy to find a set of seeds that do not yield this error.

```
workable_seeds = c(3, 6, 7, 8, 10, 11, 13, 14, 17, 22)

game.lm.MSE=rep(0,10)
for (i in 1:10){
  set.seed(workable_seeds[i])
  game.sample = sample(1:nrow(game),nrow(game)*0.8)
  game.train = game[game.sample,]
  game.test = game[-game.sample,]

  game.lm = lm(total_sales ~ console + genre + critic_score +
               release_month, data = game.train)

  game.pred = predict(game.lm, newdata=game.test)
  game.lm.MSE[i]= mean((game.pred-game.test$total_sales)^2)
}

mean(game.lm.MSE)
```

```
[1] 1.389352
```

d. Results

Here is the resulting linear regression model that best predicts `total_sales`.

```
game.lm = lm(total_sales ~ console + genre + critic_score + release_month, data = game)
summary(game.lm)
```

Call:

```
lm(formula = total_sales ~ console + genre + critic_score + release_month,
    data = game)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
```

-3.0375 -0.5482 -0.1991 0.2290 18.3364

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-1.670142	0.155199	-10.761	< 2e-16	***
consoleDC	-0.386172	0.252537	-1.529	0.126300	
consoleDS	0.138657	0.136310	1.017	0.309112	
consoleGB	0.388124	0.636328	0.610	0.541934	
consoleGBA	-0.138385	0.132040	-1.048	0.294676	
consoleGBC	0.987753	0.896637	1.102	0.270692	
consoleGC	-0.227559	0.133183	-1.709	0.087597	.
consoleGEN	-0.723496	1.261175	-0.574	0.566224	
consoleN64	-0.058213	0.187159	-0.311	0.755790	
consoleNES	4.113478	1.258587	3.268	0.001091	**
consoleNS	-0.246117	0.176709	-1.393	0.163762	
consolePC	-0.356294	0.130105	-2.738	0.006199	**
consolePS	0.343531	0.141174	2.433	0.015001	*
consolePS2	0.609687	0.132567	4.599	4.37e-06	***
consolePS3	0.617812	0.125532	4.922	8.93e-07	***
consolePS4	1.960155	0.168637	11.624	< 2e-16	***
consolePSN	0.839426	0.893135	0.940	0.347343	
consolePSP	0.071169	0.127773	0.557	0.577560	
consolePSV	0.227166	0.379074	0.599	0.549030	
consoleSAT	-0.494882	0.380226	-1.302	0.193144	
consoleSNES	-0.609630	0.894181	-0.682	0.495420	
consoleVC	-0.636034	0.893321	-0.712	0.476513	
consoleWii	0.237188	0.133687	1.774	0.076103	.
consoleWiiU	-0.220262	0.203568	-1.082	0.279313	
consoleX360	0.464805	0.122687	3.789	0.000154	***
consoleXB	-0.179864	0.164385	-1.094	0.273949	
consoleXBL	-0.587447	0.732240	-0.802	0.422449	
consoleXOne	0.734440	0.193616	3.793	0.000151	***
genreAction-Adventure	-0.322259	0.157698	-2.044	0.041065	*
genreAdventure	-0.221958	0.095906	-2.314	0.020699	*
genreBoard Game	0.223525	1.260697	0.177	0.859280	
genreEducation	-0.679285	0.888538	-0.764	0.444615	
genreFighting	-0.282198	0.098005	-2.879	0.004005	**
genreMisc	-0.139090	0.095907	-1.450	0.147064	
genreMMO	-0.948290	0.629247	-1.507	0.131882	
genreMusic	-0.467884	0.319549	-1.464	0.143217	
genreParty	0.087764	0.454436	0.193	0.846869	
genrePlatform	-0.198903	0.085904	-2.315	0.020639	*
genrePuzzle	-0.409435	0.129687	-3.157	0.001605	**

genreRacing	-0.201813	0.083046	-2.430	0.015136	*
genreRole-Playing	-0.371534	0.075939	-4.893	1.03e-06	***
genreSandbox	0.995120	1.260734	0.789	0.429972	
genreShooter	0.229148	0.072923	3.142	0.001688	**
genreSimulation	-0.036439	0.113224	-0.322	0.747596	
genreSports	-0.128283	0.072388	-1.772	0.076442	.
genreStrategy	-0.402011	0.109455	-3.673	0.000243	***
genreVisual Novel	-1.285507	1.258289	-1.022	0.307016	
critic_score	0.268355	0.014242	18.843	< 2e-16	***
release_month	0.057045	0.006005	9.500	< 2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.252 on 4074 degrees of freedom

Multiple R-squared: 0.2195, Adjusted R-squared: 0.2103

F-statistic: 23.87 on 48 and 4074 DF, p-value: < 2.2e-16

The chosen predictors are console, genre, critic_score and release_month, with critic_score and release_month being the most significant based on their p-values. The average MSE from performing cross-validation in part C is 1.39 which is not ideal. This is most likely due to the many categorical predictors in our data. They do not form a linear relationship with total_sales which is needed for Linear Regression to be effective.

3. Decision Tree

a. Model Equation

b. Process

c. Cross Validation

d. Results

4. Conclusion

As our previous analyses on our game data has shown, the _____ model yielded the lowest MSE with a value of _____, making it the most effective at computing `total_sales`. This is most likely due to _____.

These findings will be valuable to game developers to get a general idea of how successful their game will be. It will also provide insight into the general audience's interest into what type of games they enjoy, and what factors they consider before purchasing a game.

5. Bibliography

- Data set: <https://www.kaggle.com/datasets/hosammhmdali/video-game-sales-2024>

6. Source Code

a. Code for Data Preparation//Tuan Pham

```
#include necessary library
library(tidyverse)
library(lubridate)
library(dplyr)
#extract and get the predictors column that we are interested in
game <- game %>% select(c("genre", "publisher", "developer", "critic_score", "total_sales", "release_date"))
#extract and get the month from release_date
game$release_month <- month(game$release_date)
#drop the missing values and last unnecessary column
game <- game %>% select(-c("release_date"))
game <- na.omit(game)
#change categorical variables' data type from text to factor for easier processing
game$console <- factor(game$console)
game$genre <- factor(game$genre)
game$publisher <- factor(game$publisher)
game$developer <- factor(game$developer)
```

b. Code for Linear Regression //Uyen Vi Phan

```
library(readr)
game <- read_csv("game.csv")
#View(game)
attach(game)

#check how many categories in each column
categories <- unique(game$console)
length(categories)
categories <- unique(game$genre)
length(categories)
categories <- unique(game$publisher)
length(categories)
categories <- unique(game$developer)
length(categories)
```

```

categories <- unique(game$console)
length(categories)

#view frequency of each column
game.pub.types = table(game$publisher)
#View(game.pub.types)
game.dev.types = table(game$developer)
#View(game.dev.types)
game.gen.types = table(game$genre)
#View(game.gen.types)
game.con.types = table(game$console)
#View(game.con.types)

game.lm = lm(total_sales ~ console+genre+critic_score+release_month, data = game)
summary(game.lm) #cant use t-test bc of the categorical vars

step(game.lm) #backwards stepwise regression

set.seed(22) #keep incrementing until find good seed
game.sample = sample(1:nrow(game),nrow(game)*0.8)
game.train = game[game.sample,]
game.test = game[-game.sample,]

game.lm = lm(total_sales ~ console+genre+critic_score+release_month, data = game)

game.pred = predict(game.lm, newdata = game.test)
game.lm.MSE= mean((game.pred-game.test$total_sales)^2)

workable_seeds = c(3, 6, 7, 8, 10, 11, 13, 14, 17, 22)

game.lm.MSE=rep(0,10)
for (i in 1:10){
  set.seed(workable_seeds[i])
  game.sample = sample(1:nrow(game),nrow(game)*0.8)
  game.train = game[game.sample,]
  game.test = game[-game.sample,]

  game.lm = lm(total_sales ~ console + genre + critic_score +
               release_month, data = game.train)

```

```
game.pred = predict(game.lm, newdata=game.test)
game.lm.MSE[i]= mean((game.pred-game.test$total_sales)^2)
}

game.lm.MSE
```

c. Code for Regression Tree//Adeer Siddiqui