Data Engineering Final

DATA ENGENEERING ON GAMING INDUSTRY DATASETS

Zan Aleksander Bozic

Hult International Buisness School

Abstract

In this paper I explored three different data sets to see if there can be a correlation drawn between successful video games and whether they have launched on a corresponding successful platform. To analyse the data, I first look at related work done on the topic, further I show the importance and methods used in normalisation of the three data sets, and I draw my conclusion through my final analysis in the last section of the paper. I use three different tools for my data analysis. I use a MySQL DBMS called POPSQL, furthermore I use snippets of Python code developed in PyCharm. all the databases that I use are imported and processed locally in MySQL. To see the full project please click on this link to access the GIT repository: https://github.com/zbozic18/gaming_data_engeneering.

Table of Contents

Introduction:

There has been an ongoing feud between companies which are producing different gaming consoles. This feud has taken many battles throughout the past 30 years. (Sterman, 2011) Many of these battles involve produces or gaming consoles coming from Japan. This paper will focus whether gaming producers need to be mindful of these battles by finding the level of correlation between the success of a gaming console and the probability of a video game succeeding if it launches on that respective console. This is important because if the correlation is low, then it means that the feud between gaming consoles is strictly limited to the console market and the rest of the gaming industry should not pay much attention to it.

To complete firm conclusion, this analysis will firstly look at the related work that has already been done on some of the data sets also used in further analysis. Furthermore, the paper will look at the analysis that can be done on specific data sets which look at video game sales, console sales, and popularity of specific video games. Most of the analysis done on dimension data sets has been conducted in a form of MySQL query analysis. Finally, the paper will also aim to criticise the data sets and the approaches used in this paper. This is to an end to suggest further analysis that can be built upon the limitations of this assignment.

Hypothesis:

The number of console sales has a positive effect on video games sales launched on the respective consoles.

Related Work:

There is a selection of related work to analyse before diving into the actual analysis of the data sets. The aim of analysing related work is to see what other work has been already performed, to see which assumptions have already been proven or disproven. By looking at figure one, we can see that the market of video games has been growing strongly over the past 30 years. This confirms the relevancy of understanding this hypothesis, as it means that there is a lot of interest of different investors, contenders, and consumers to understand the relationship between consoles and video games.

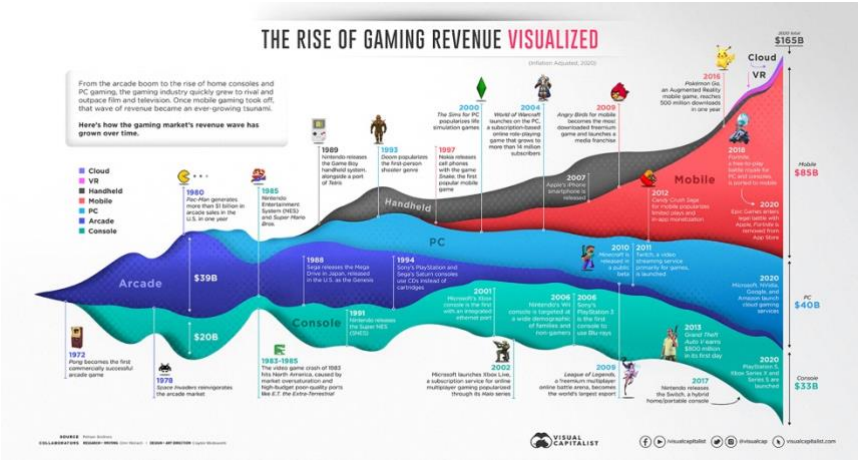

Figure 1: (Wallach, 2020)

Furthermore, by looking at figure 2, we can see a large disproportion between the number of games available on PC and number of games available on other consoles. Therefore, to avoid this bias this research does not include PC as a gaming platform in its analysis. Additionally, it is worth noticing how PlayStation 4 and Xbox 360, which are considered one the two closest competitors throughout the battle between consoles, are the two leaders in the number of games by platform.

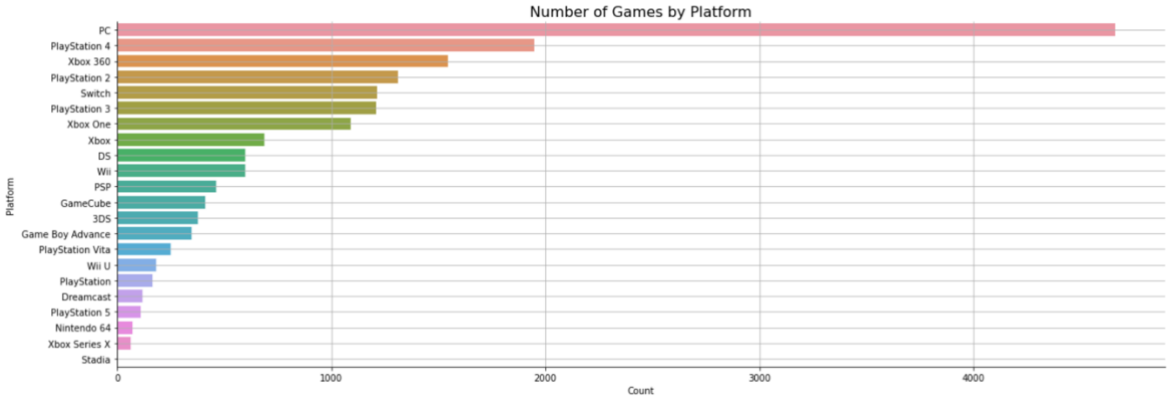

Figure 2: (Video Game - Data Visualization, n.d.))

There has also been some analysis already done in terms of which platforms have leads in the past 30 years. In figure 3, we can see a visual graphic showing top platforms by revenue for each year. We can see that between the years of 1996 and 1016, when platform sales have really reached peaks, the main leaders are different series of PlayStations, we and Xbox 360. This already gives an indication in terms of what kind of findings we will get throughout further analysis of data sets.

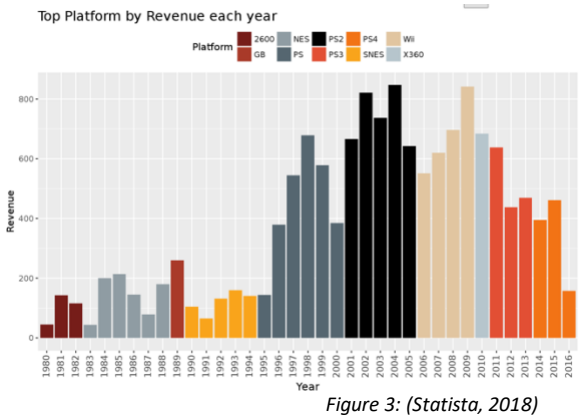

Figure 3: (Statista, 2018)

Data Sets

There are three different data sets which are used in the further analysis through MySQL queries. All the three data sets have two matching fields which allow us a cross analysis between them. These two matching fields are: platform and years.

Data Set 1: Platform Sales

Platform sales data set is a data set downloaded from Statista. It's a small data set which shows the volume sales, in million units per platform for the range of years between 2004 and 2020. It covers the historical sales for 13 different consoles. Each console represents one column in the data set. Besides the 13 columns of the console's, there is an additional column which lists all the years. Figure 4 shows how the data is seen in an excel sheet. As seen in the table there are 17 records in this data set.

Years	Wii U	DS	Wii	Xbox 360	Playstation V	Playstation 3	PSP	3DS	Xbox Series X	Xbox One	Playstation 5	Playstation 4	Switch
2004		2.86					0.47						
2005		10.99		1.18			9.61						
2006		20.76	2.96	6.81		1.25	9.49						
2007		29.31	16.55	7.91		7.92	12.81						
2008		29.47	24.09	11.16		10.46	14.05						
2009		27.28	21.05	10.36		13.26	9.92						
2010		20.55	17.26	13.54		13.83	9.15						
2011		8.76	11.49	13.95	0.48	14.42	7.52	12.48					
2012	2.16	3.08	5.11	10.69	3.6	11.97	4.24	13.28					
2013	3.07	0.96	2.04	6.24	3.36	8.26	2.98	13.95		3.06		4.46	
2014	3.46		0.79	2.54	2.66	3.44	0.57	8.88		7.23		13.73	
2015	3.58		0.3	0.99	2.76	1.72		7.83		8.39		17.74	
2016	1.16			0.39	2	0.67		7.3		7.98		17.79	
2017	0.13			0.06	0.78	0.2		6.64		7.64		19.81	13.12
2018					0.23			3.54		6.83		18.28	16.34
2019					0.03			1.55		4.97		14.27	19.28
2020								0.46	3.03	3.15	4.39	8.6	28.3

Figure 4

Data Set 2: Video Games Sales

Data set 2 is a larger data set then data set 1, which encompasses a total of 11493 records. The data set shows a record of video game sales per video game. It consists out of the following columns show in table 1. The key columns, from this data set, in this analysis, are *Platform*, *Year, NA_Sales, EU_Sales, JP_Sales,* and *Other_Sales*.

Rank	*Name*	***Platform***	***Year***	*Genre*	*Publisher*
Primary Key in the data; unique identifier.	Video game name.	Console name	Year of release.	Genre of the game.	Publishing company of the game.

NA_Sales	***EU_Sales***	***JP_Sales***	***Other_Sales***
Sales volume in the first year of game launch in the North Americas region.	Sales volume in the first year of game launch in the European region.	Sales volume in the first year of game launch in Japan.	Sales volume in the first year of game launch in countries apart from NA, Europe and Japan.

Table 1

Data Set 3: Video Game Scores

Data set number 3 explores the video game scores in terms of popularity. There are two main scores which are assessed: the meta score (the rank which was given by the critics) and the user review score (which is based on user reviews). The data set has a total of 12254 entries. It consists of the columns show in table 2. Columns *release_date, platform,* and *user_review* are the main columns which are used in the analysis. It is worth mentioning that the meta score column would be ignored, as the user review will take the main measurement for the game rating. This assumption is based on the explanation that users are the end customers and therefore it is a more appropriate metric in this analysis.

name	***platform***	***release_date***	*summary*	*meta_score*	***user_review***
Video game name.	Console name.	Month and year when the video game was released.	Description of the game.	Meta score of the game.	Average user review of the game.

Table 2

Methods:

The analysis will be based on three main steps. The first step will be moving the data from the CSV files in two a locally hosted MySQL database. Second step will be the normalisation of data. The last step of the process is the analysis of the data with the aim to prove the set hypothesis.

Step 1:

Step 1 had an aim to create three normalised tables, which allow the user to

Because of the formatting of specific columns within each data set, a simple table import wizard from MySQL workbench cannot be used. Table three shows all of the identified issues which had to be solved in order to move the data from the CSV files into a MySQL database.

Data Set	Issues
Data Set 1: Platform Sales	- Empty values instead of '0.0' → Does not allow some mathematical queries in sql to work. - Columns are the consoles, which can present inconvenience issue with matching queries in the analysis.
Data Set 2: Video Games Sales	- 6 unexplained columns. - 'N/A' value in some Years columns. - 'tbd' as a value in some columns.

	- Random values of 2600 and 3600 reappearing in the Platform column - A random sequence of 4 numbers appearing in appearing in the Name column.
Data Set 3: Video Game Scores	- Description column has “**,**” in the descriptions and is not string separated with “**””**. This causes an error when importing the csv file into a MySQL database.

Table 3

Fixing & Importing Data Set 1:

The first code snippet shown in figure 5, displays a function fix_empty_values(). This function philtres through console sales CSV file and replaces all of the empty values with a value of ‘0.0’. This now enables us to further use the 0.0 values in MySQL mathematical functions later in the analysis. Furthermore the function calls a second function called reverse_console_sales(), which refers to the function shown in figure six. This function uses the pandas library to switch the column names with values in the column Years. This enables us more effective querying later in MySQL analysis.

After the two functions have normalised the CSV files, we use the code snippet shown in figure 7 to write all the rows into the console_s table into the MySQL database.

```python
# This function fills all the empty value in the csv with 0.0 in order to enable mysql operations
# It does so by checking every value in every row of the Console_Sales.csv for missing values, substituting them w/ 0.0
# It appends all the new rows onto a new list and writes them onto a new csv file: 'console_sales_fixed.csv'
# Finally it calls the function reverse_console_sales() which saves the amended data onto a new csv file
def fix_empty_values():
    fixed_data = []
    with open('Console_Sales.csv', 'r') as file:
        data = csv.reader(file)
        first = True
        for row in data:
            if not first:
                fixed_row = []
                for i in row:
                    if i == '':
                        fixed_row.append(0.0)
                    else:
                        fixed_row.append(i)
                fixed_data.append(fixed_row)
            elif first:
                print(row)
                first_row = row
                first = False

    with open('console_sales_fixed.csv', 'w') as file:
        csvw = csv.writer(file)
        csvw.writerow(first_row)
        csvw.writerows(fixed_data)

    reverse_console_sales()
```

Figure 5

```python
    # This function saves an entry in the games database, console_s table. It requires input for all the columns.
    def save_row_console_sales(self, platform, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15, y16,
                               y17
                               ):
        try:
            self.cursor.execute(
                """
            INSERT INTO console_s(
            platform, `2004`, `2005`, `2006`, `2007`, `2008`, `2009`, `2010`, `2011`, `2012`, `2013`, `2014`, `2015`,
            `2016`, `2017`, `2018`, `2019`, `2020`
            )
            VALUES ('{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}',
            '{}', '{}')
            """.format(platform, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15, y16, y17)
            )
            self.db_connection.commit()
        except mysql.Error as e:
            print(e)
```

Figure 6

```python
# This function uses the pandas data frame in order to rotate the columns and the rows of the csv file.
# This makes the columns of the data set switch from consoles to years for easier mysql querrying.
def reverse_console_sales():
    sales = pd.read_csv('console_sales_fixed.csv')
    print(sales)
    sales = sales.T
    print(sales)
    sales.to_csv('console_s.csv')
```

Figure 7

Fixing & Importing Data Set 2:

Solving data sets to is done by filtering every single row in terms of the values it should not possess (shown in table 3), to get a new list of rows which do not include the ones with the incorrect values. Once the rows have been filtered a new CSV file is created including only the correct rows. The underlying function for this process can be seen in figure 8. Once a fixed CSV has been generated, we use the code snippet in Figure 9 to save the data into the table in the MySQL database.

```
# Fixes the vg_sales file by filtering out the entries which fall into one of the following categories:
# year = 'N/A', critic_count = 'tbd', platform = '2600' or '3600', len(name) = 4
# The function appends all of the filtered rows on a new list and saves it on a a new csv document: 'vg_sales_fixed.csv
def fix_vg_sales():
    new_rows = []
    names = ['Name', 'Platform', 'Year_of_Release', 'Genre', 'Publisher', 'NA_Sales', 'EU_Sales', 'JP_Sales',
             'Other_Sales', 'Global_Sales', 'Critic_Score', 'Critic_Count', 'User_Score', 'User_Count', 'Developer',
             'Rating'
             ]
    with open('vg_sales.csv', 'r') as file:
        data = csv.reader(file)
        starter = True
        for row in data:
            if not starter:
                year = row[2]
                if year != 'N/A':
                    if row[12] != 'tbd':
                        if row[1] != 2600:
                            if row[1] != 3600:
                                if len(row[0]) != 4:
                                    new_rows.append(row)
            starter = False

    with open('vg_sales_fixed.csv', 'w') as file:
        csvw = csv.writer(file)
        csvw.writerow(names)
        csvw.writerows(new_rows)
```

Figure 8

```
    # This function saves an entry in the games database, console_s table. It requires input for all the columns.
    def save_row_console_sales(self, platform, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15, y16,
                               y17
                               ):
        try:
            self.cursor.execute(
                """
                INSERT INTO console_s(
                platform, `2004`, `2005`, `2006`, `2007`, `2008`, `2009`, `2010`, `2011`, `2012`, `2013`, `2014`, `2015`,
                `2016`, `2017`, `2018`, `2019`, `2020`
                )
                VALUES ('{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}', '{0}',
                '{0}', '{0}')
                """.format(platform, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15, y16, y17)
            )
            self.db_connection.commit()
        except mysql.Error as e:
            print(e)
```

Figure 9

Fixing & Importing Data Set 3:

To fix data set 3, I simply deleted the column descriptions and use the data table import wizard.

Normalisation in MySQL:

Before starting a concrete analysis in MySQL, I had to normalise the platform column which bridges the three tables. Thus, I looked at which platforms are present in the given data sets. I found that there was a mismatch between the available platforms in each data set. The result is shown in Figure 10. Additionally, we can see that there is a mismatch between the naming conventions. This was easily solved by deleting rows which included

platforms that were not present in all three. To do so I used **DELETE FROM** and **WHERE** commands. Lastly game reviews had a wide space in front of every single string which had to be fixed. To do so I used **UPDATE, SET** and **WHERE** commands.

platform	platform	platform
3DS	3DS	Wii U
DC	Dreamcast	DS
DS	DS	Wii
GBA	Game Boy Advance	Xbox 360
GC	GameCube	Playstation Vita
PC	Nintendo 64	Playstation 3
PS2	PC	PSP
PS3	PlayStation	3DS
PS4	PlayStation 2	Xbox Series X
PSP	PlayStation 3	Xbox One
PSV	PlayStation 4	Playstation 5
Wii	PlayStation 5	Playstation 4
WiiU	PlayStation Vita	Switch
X360	PSP	
XB	Stadia	
XOne	Switch	
	Wii	
	Wii U	
	Xbox	
	Xbox 360	
	Xbox One	
	Xbox Series X	

Figure 10

Proving They Hypothesis

Once all of the normalisation has been concluded, the experiment on the hypothesis was quite straight forward. I started by choosing four years within the range of 2004 in 2020, for which I will test the hypothesis in. The selected years war 2004, 2008, 2012, and 2016.

I started by finding the top console in terms of sales volumes for the selected four years. for this I used the script shown in figure 11. Table 4 shows the results.

```
# Find the top consoles for 4 years

SELECT platform, `2004` FROM console_s
ORDER BY `2004` DESC
LIMIT 1; # DS: 2.86

SELECT platform, `2008` FROM console_s
ORDER BY `2008` DESC
LIMIT 1; # DS: 29.47

SELECT platform, `2012` FROM console_s
ORDER BY `2012` DESC
LIMIT 1; # 3DS: 13.28

SELECT platform, `2016` FROM console_s
ORDER BY `2016` DESC
LIMIT 1; # PlayStation 4 17.79
```

Figure 11

Year	Platform	Sales
2004	DS	2.86m
2008	DS	29.47m
2012	3DS	13.28m
2016	PlayStation4	17.79m

Table 4

After I found the top selling platforms for each year, I wanted to find the video games and the sales of the video games in respect to the platform and the years in order to draw the correlation. Therefore, I used the query shown in figure 12 using inner join. the query finds all the game names come up platforms, release years and total sales for each game in table vgsales and joins them with the platform field from the console_s table. However, there was a problem with this query as it returned 2099 rows.

```
# Find the video games and the the sales of the games in respect to the platform
#and years

SELECT vgsales.game_name,
vgsales.platform,
vgsales.release_year,
(vgsales.na_sales + vgsales.eu_sales + vgsales.jp_sales + vgsales.global_sales)
as total_sales
FROM vgsales
INNER JOIN console_s ON console_s.platform=vgsales.platform
WHERE
vgsales.release_year = '2004' OR
vgsales.release_year = '2008' OR
vgsales.release_year = '2012' OR
vgsales.release_year = '2016'
ORDER BY release_year ASC, na_sales DESC;
```

Figure 12

To create a more general query, I try to find the average game sales per platform for a given year. This can be shown in figure 12 where I used a select statement to find the platform the average sales and the standard deviation off sales for a given year. The statement groups the results by platform and orders them by the average sales and descending order. I repeated the same query for every one of the four selected years. The results are shown in table 5. From the results in the year 2004 and 1016, our correlation is proven. However, the years thousand 18,012 the top consoles only showed up as fifth. This result further questions the hypothesis. Especially when trying to draw correlation the standard deviations which represent a high percentage of the averages, are too large. Therefore, this test disproves the hypothesis.

```
# Find the average game sales for the years and platforms

SELECT platform,
ROUND((AVG(na_sales) + AVG(eu_sales) + AVG(jp_sales) + AVG(other_sales)), 2)
AS avg_sales,
ROUND(((STD(na_sales) + STD(eu_sales) + STD(jp_sales) + STD(other_sales)) / 4), 2)
AS std_sales
FROM vgsales
WHERE release_year = '2004'
GROUP BY platform
ORDER BY avg_sales DESC; # DS is 1
```

Figure 12

Year	Result
2004	- DS is first (sales: 0.76mil, std: 0.55)
2008	- Xbox is first (sales: 0.93mil, std: 0.38) - DS is fifth (sales: 0.3, std: 0.19)
2012	- Xbox is first (sales: 0.95mil, std: 0.47) - 3DS is fifth (sales: 0.55mil, std: 0.37)
2016	- PlayStation 4 (sales: 0.37mil, std: 0.22)

Table 5

Conclusion:

Based on the final analysis the hypothesis can be rejected. There is no concrete evidence shown from the actual analysis which can draw a link between the correlation of high success in platform and the high success in video game launched that platform. However, it has to be noted that the analysis conducted in this assignment has been quite limited period in order to explore this correlation further come out we would need to assess all of the years not just the four selected. What's is also important is that this approach conducted through this analysis shows how strong normalisation in the beginning of the analysis creates solid grounds for effective data exploration. Therefore, always when analysing data, we should start by a strong normalisation structure which gives solid grounds for further data analysis.

Table of used queries:

Table 6 shows a list of used queries in the full MySQL script.

Command
CREATE TABLE
SELECT
SHOW tables
DESCRIBE
LIMIT
COUNT
MAX/MIN
UNION
DISTINCT
COUNT
DELETE
WHERE
UPDATE SET
INNER JOIN
ORDER BY (ASC/DSC)
ROUD

Table 6

References:

1. Explore Video Games Sales. (n.d.). Kaggle.com. https://www.kaggle.com/umeshnarayanappa/explore-video-games-sales/report
2. Katzenbach, C., Herweg, S., & Roessel, L. van. (2016). Copies, Clones, and Genre Building: Discourses on Imitation and Innovation in Digital Games. International Journal of Communication, 10(0), 22. https://ijoc.org/index.php/ijoc/article/view/4802/1567
3. Statista (2018). Video gaming console market share 2018 | Statista. [online] Statista. Available at: https://www.statista.com/statistics/276768/global-unit-sales-of-video-game-consoles/.
4. Sterman J. (2011). Sony's Battle for Video Game Supremacy. MIT Sloan Management. Available at: http://dln.jaipuria.ac.in:8080/jspui/bitstream/123456789/2811/1/Sony%27s%20Battle%20for%20Video%20Game%20Supremacy%20.pdf
5. Video Game - Data Visualization. (n.d.). Kaggle.com. Retrieved December 18, 2021, from https://www.kaggle.com/mayurdalvi/video-game-data-visualization
6. Wallach, O. (2020, November 23). 50 Years of Gaming History, by Revenue Stream (1970-2020). Visual Capitalist. https://www.visualcapitalist.com/50-years-gaming-history-revenue-stream/