

Project 3: Regression Modeling of Video Game Sales

Karim Sasa, 11/01/2024

Kaggle Dataset:

<https://www.kaggle.com/datasets/asaniczka/video-game-sales-2024>

Introduction to the Problem:

I'm a big fan of video games, and I spend a good chunk of my time exploring different titles across various different consoles and genres. Everyone has their favorite games, but what really drives a game's global sales beyond just being fun to play? Even though I love gaming, I'm not exactly sure what factors contribute to a game's worldwide success. So, I decided to dive into this Kaggle dataset that includes info on video games like console, genre, critic score, and developer. I want to figure out how these features impact global sales and see which ones make the biggest difference.

Introduction to the Data:

The data comes from a Kaggle dataset, and it gathers info from a large number of video games along with their global sales numbers. There are several features listed for each game, including the image, game title, console, genre, publisher, developer, critic score, total sales, and regional sales like North America, Japan, Europe/Africa, and others. There's also data on the release date and when the data was last updated. All these features speak for themselves, so there's no need to describe each one.

What is Regression and How Does It Work?:

To start, I'm going to explain how I answer my questions using linear regression. Regression analysis is all about understanding the relationship between a dependent variable and one or more independent variables. In my case, the dependent variable is the global sales of video games, and the independent variables are things like console, genre, critic score, and developer.

Linear Regression in a Nutshell

To go more in depth about linear regression, it is one of the simplest and most commonly used types of regression analysis. It's able to model the relationship between the dependent variable and independent variables by fitting a linear equation to observed data. The basic idea is to find the best-fitting straight line through the data points.

The mathematical representation of a simple linear regression (with **one** independent variable) is:

$$y = \beta_0 + \beta_1 x + \epsilon$$

But since I have multiple features, I'll be using multiple linear regression:

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

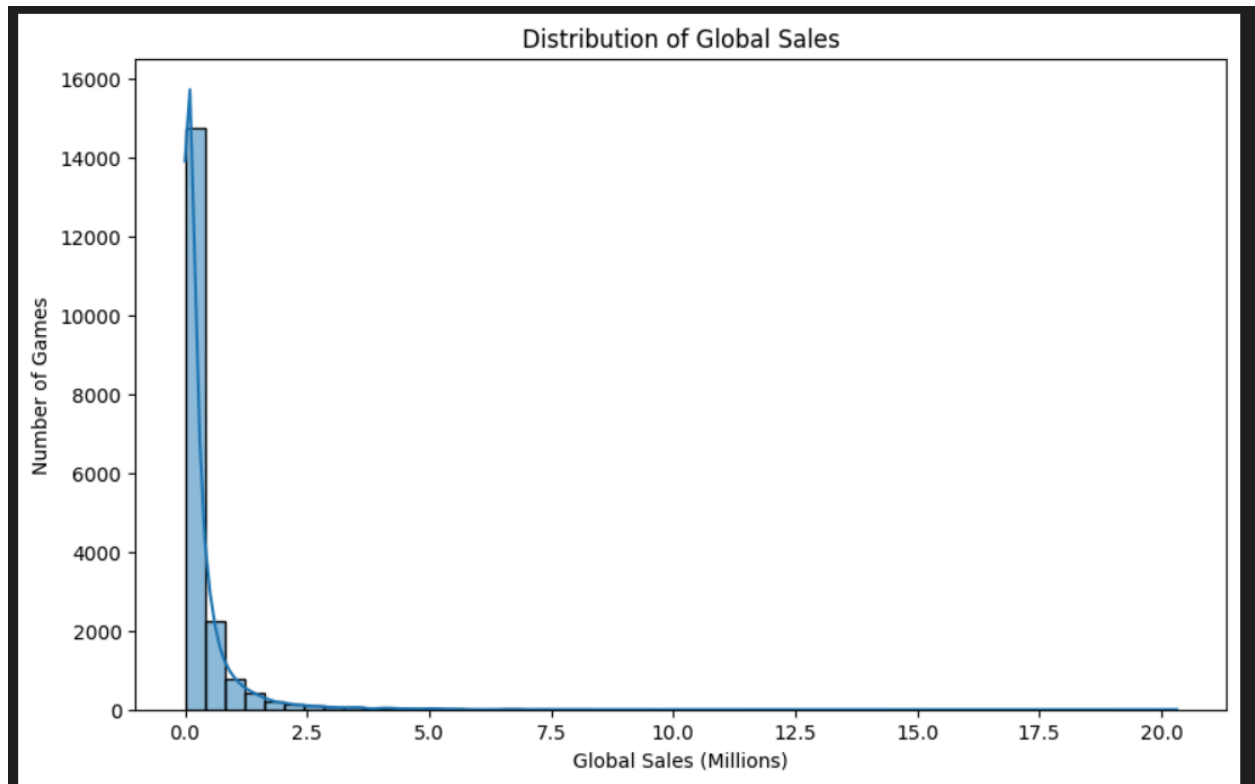
Where:

- **y** is the dependent variable (global sales).
- **x_1, x_2, \dots, x_n** are the independent variables (console, genre, critic score, developer, etc.).
- **β_0** is the intercept (the expected mean value of y when all $x_i = 0$).
- **$\beta_1, \beta_2, \dots, \beta_n$** are the coefficients representing the impact of each independent variable on y.
- **ϵ** is the error term (the difference between the predicted and actual values).

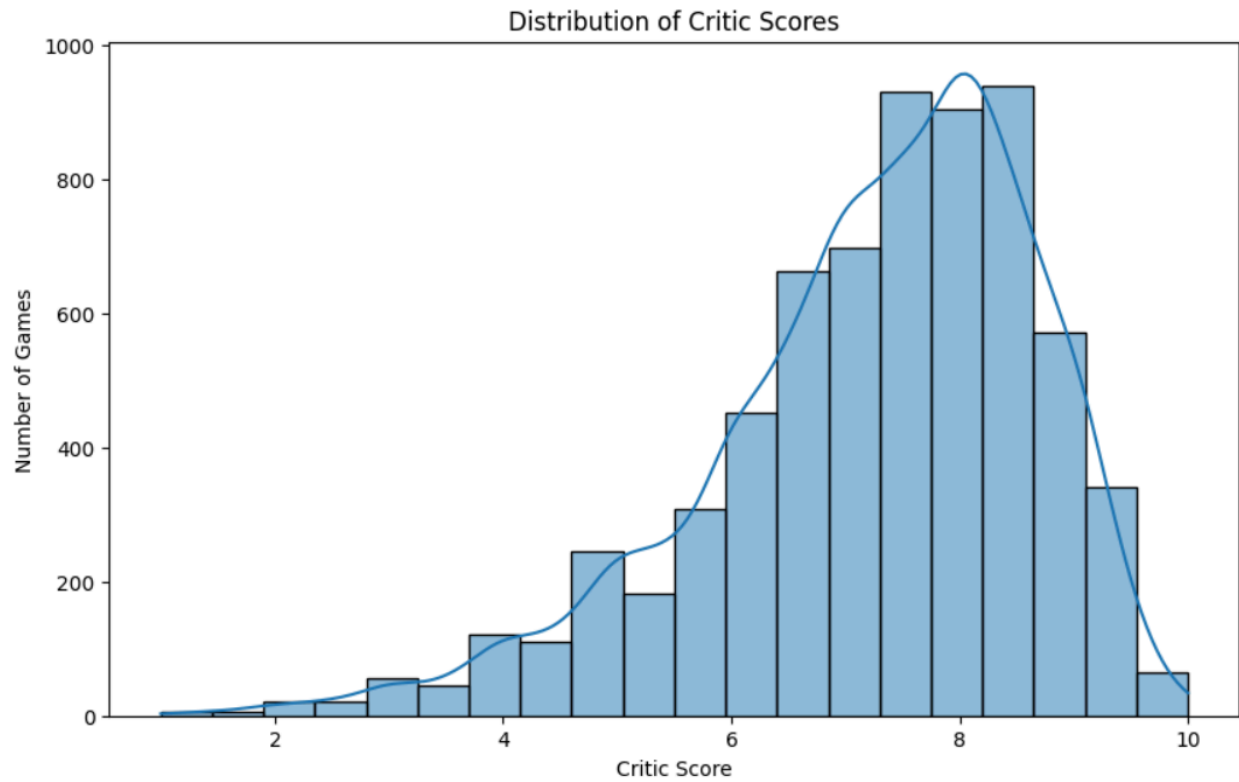
Experiment 1 - Data Understanding:

Before I go into building the regression model, it's important to get an understanding of the data I'm working with. By understanding the dataset it will help me to identify any potential issues, uncover patterns, and determine the best way to approach the analysis.

First I wanted to see any relations between the total sales, and the critic scores. First, I made a visualization of the distribution of total sales:

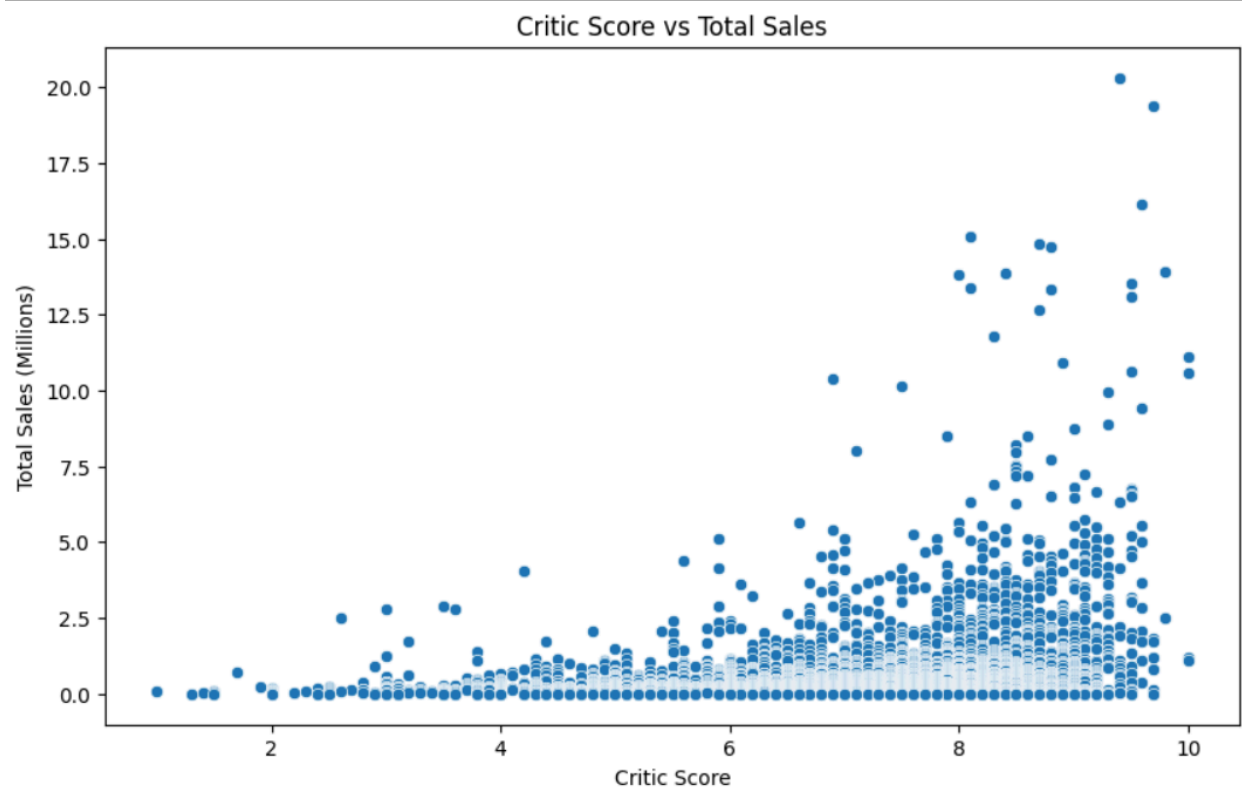


The histogram shows that most games sell fewer than about 1 million copies, with a few outliers selling over 2.5 million.



Most games have critic scores between 6 and 8, which seems about right.

Now I'd like to explore some relations between both these features by using a scatter plot:



Now, we can see there are definitely some patterns between a higher critic score, and higher game sales. Also, a lower critic score contributes to a lower game sale.

Experiment 1 - Pre-processing step:

For this experiment, the feature I'm focused on is the `critic_score` feature, and how it impacts global sales. I'm starting with this simplified approach, as it first allows me to explore the relationship between a game's critical reception and its overall sales performance. Since `critic_score`, and `total_sales` have several missing values, I decided to handle them by filling the values with the median to prevent outliers.

So in this case:

Independent Variable: `critic_score`

Dependent Variable: `total_sales`

Though I'm mainly focusing on the preprocessing for critic_score total_sales, I'm still going to drop unnecessary features like the image column, as that won't contribute to this project, as well as the title column.

I'm only going to focus on the nulls of the features of **this experiment**, so we are filling both total_sale, and critic_score with median values.

Before

```
title          0
console        0
genre          0
publisher      0
developer      17
critic_score   57338
total_sales    45094
na_sales       51379
jp_sales       57290
pal_sales      51192
other_sales    48888
release_date   7051
last_update    46137
dtype: int64
```

After

```
img          0
title        0
console      0
genre        0
publisher    0
developer    17
critic_score 0
total_sales  0
na_sales     51379
jp_sales     57290
pal_sales    51192
other_sales  48888
release_date 7051
last_update  46137
dtype: int64
```

Next I checked for duplicate values:

Duplicates

```
# Checking for Duplicated Data
duplicates_irrelevant = game_df[game_df.duplicated()]

duplicates_irrelevant
✓ 0.0s
```

	console	genre	publisher	developer	critic_score	total_sales	na_sales	jp_sales	pal_sales	other_sales	release_date	last_update
2439	DS	Misc	IE Institute	IE Institute	7.5	0.66	NaN	0.66	NaN	NaN	2006-11-09	NaN
11689	N64	Sports	Nintendo	HAL Laboratory	7.5	0.07	NaN	0.07	NaN	NaN	2000-03-31	2018-08-30
12821	DS	Misc	Nintendo	TOSE	7.5	0.05	NaN	0.05	NaN	NaN	2006-04-20	NaN
14477	DS	Adventure	Level 5	Level 5	7.5	0.03	NaN	0.03	NaN	NaN	2009-06-18	NaN
14754	Wii	Racing	Unknown	Brain in a Jar	7.5	0.03	0.03	NaN	0.0	0.0	2011-03-15	NaN
...
63711	And	Strategy	Unknown	Team17	7.5	0.12	NaN	NaN	NaN	NaN	NaN	2021-01-26
63717	PC	Strategy	Unknown	Team17	7.5	0.12	NaN	NaN	NaN	NaN	NaN	2021-01-26
63730	And	Strategy	Unknown	Konami	7.5	0.12	NaN	NaN	NaN	NaN	NaN	2022-10-12
63779	X360	Visual Novel	5pb	5pb. Games	7.5	0.12	NaN	NaN	NaN	NaN	2011-10-27	2018-12-19
63844	PC	Visual Novel	Entergram	Entergram	7.5	0.12	NaN	NaN	NaN	NaN	2018-08-23	2018-09-26

1940 rows x 13 columns

However, I will not be dropping these duplicates as developers develop different games, same goes for publishers, and critic scores can also be the same value as well. So, dropping duplicates is not in my best interest.

Both critic_score and total_sales are already in numerical form, so there is no need for type conversion. Now for syntax errors, there are no typos or whitespace. The

standardization step is not required as all the data types converted are lowercase already lowercase.

This is the current preprocessed data frame; however, the preprocessing is not over:

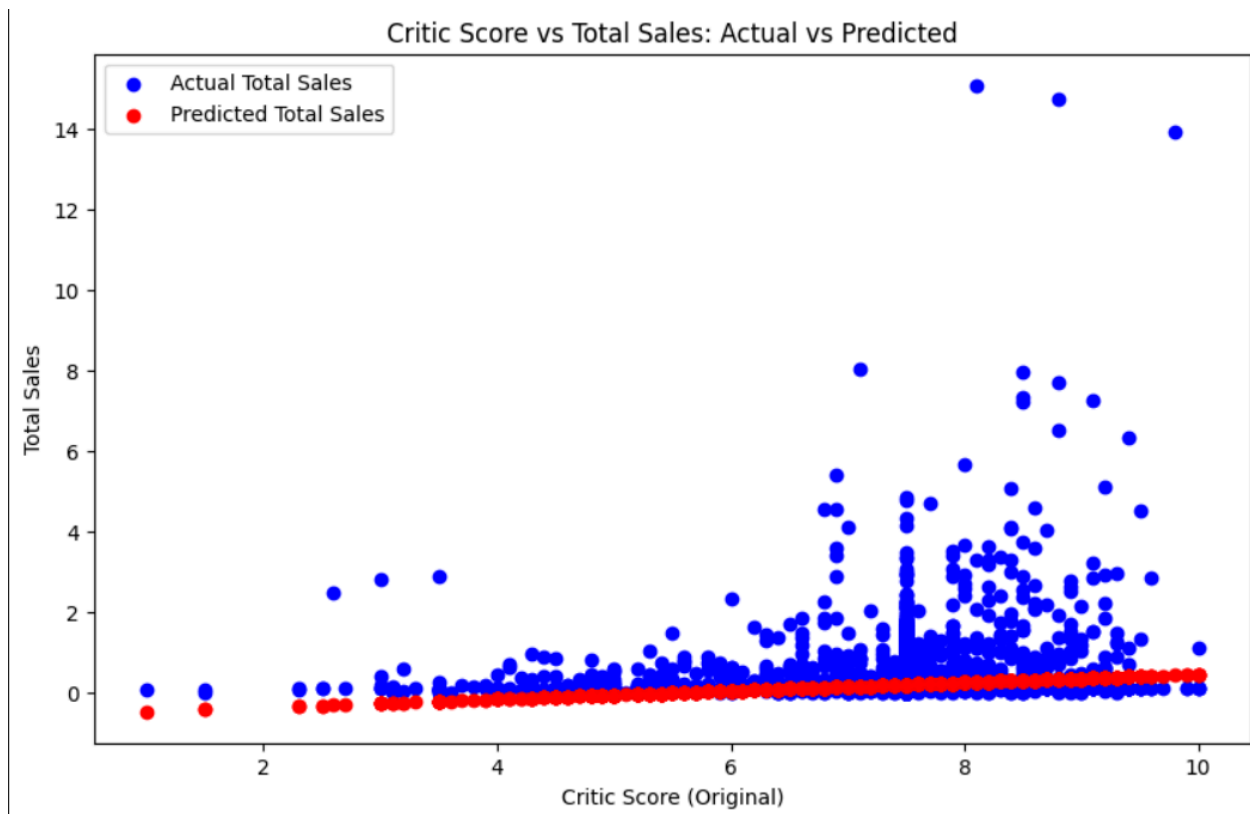
```
df_cleaned.head()
```

	console	genre	publisher	developer	critic_score	total_sales	na_sales	jp_sales	pal_sales	other_sales	release_date	last_update
0	PS3	Action	Rockstar Games	Rockstar North	9.4	20.32	6.37	0.99	9.85	3.12	2013-09-17	NaN
1	PS4	Action	Rockstar Games	Rockstar North	9.7	19.39	6.06	0.60	9.71	3.02	2014-11-18	2018-01-03
2	PS2	Action	Rockstar Games	Rockstar North	9.6	16.15	8.41	0.47	5.49	1.78	2002-10-28	NaN
3	X360	Action	Rockstar Games	Rockstar North	7.5	15.86	9.06	0.06	5.33	1.42	2013-09-17	NaN
4	PS4	Shooter	Activision	Treyarch	8.1	15.09	6.18	0.41	6.05	2.44	2015-11-06	2018-01-14

Experiment 1 - Modeling:

I split the dataset into training and testing sets (80% training, 20% testing) and trained a linear regression by using Scikit-Learn.

Here's a visualization of the linear regression model trained on the dataset:



With this visualization, we can see that while critic score may contribute slightly to predicting total sales, it's definitely not enough on its own. The model's inability to

capture the wide range of actual sales shows that additional features (such as console, genre, or publisher) should be included in my future experiments to improve the accuracy of predictions.

Experiment 1 - Evaluation:

```
Mean Squared Error (MSE): 0.18267302942792907  
Root Mean Squared Error (RMSE): 0.4274026549144601  
R-squared: 0.009115730055546
```

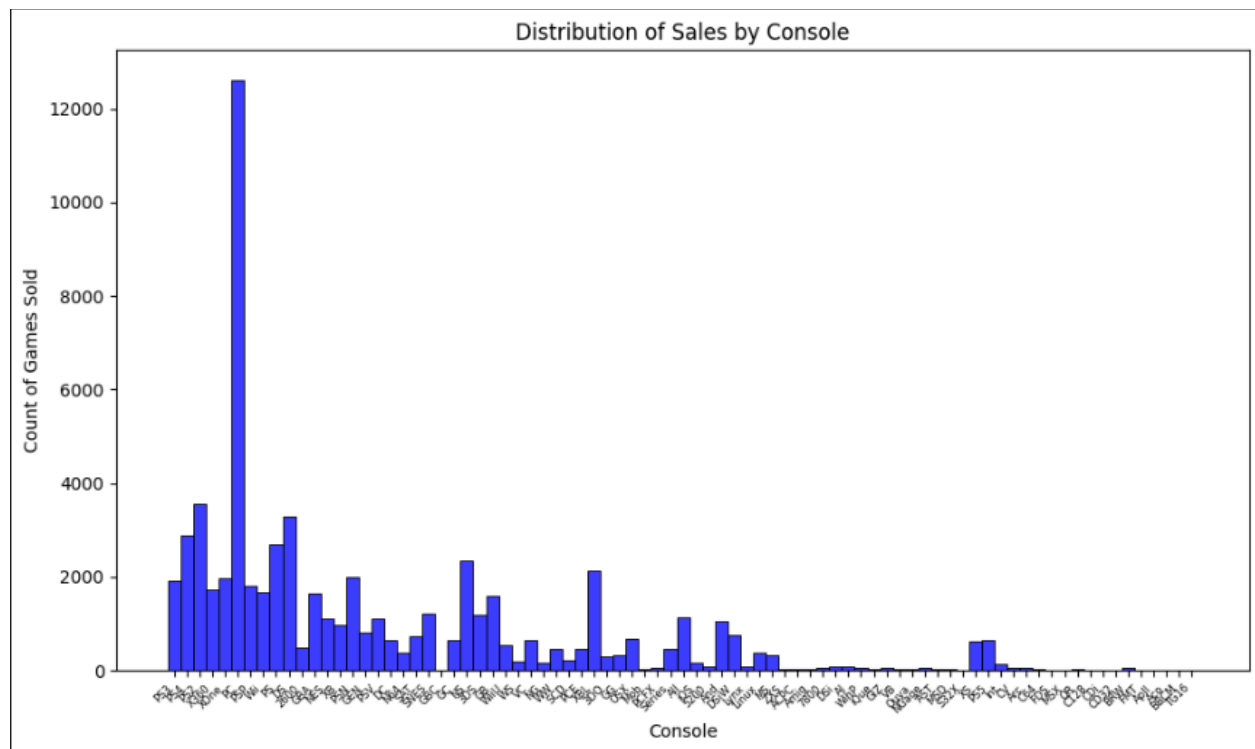
Lets focus on root mean squared error:

The Root Mean Squared Error was 0.4274, which is the square root of the MSE. In simple terms, this means that the model's predictions, on average, deviate by about .43 units from the actual total_sales values. This value shows that while the predictions are close, they aren't perfectly aligned with the actual values.

Experiment 2 - Data Understanding:

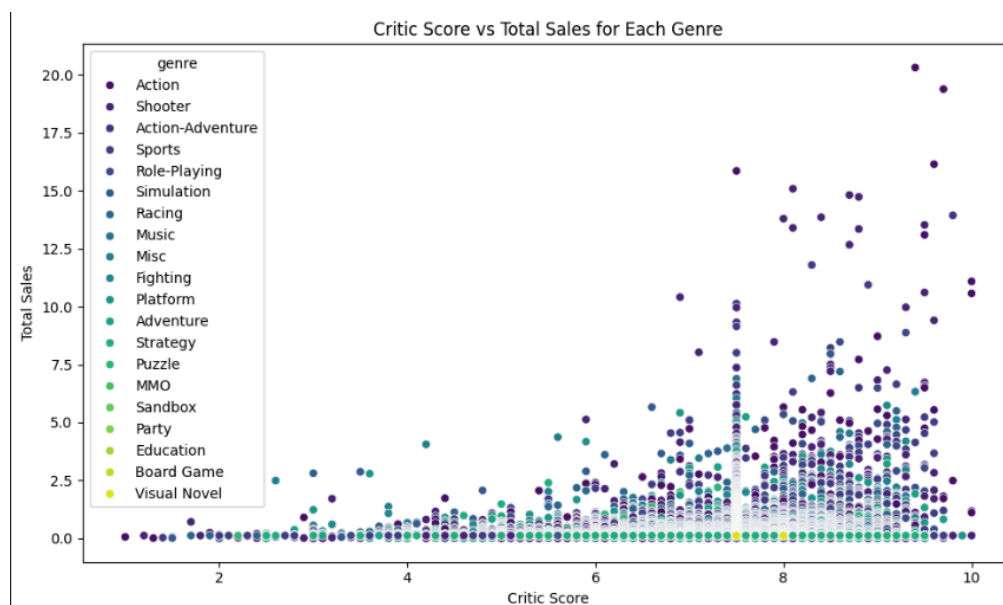
Now moving onto experiment 2, I expanded the feature set to include additional variables that might have more of an impact on predicting the total sales. For this experiment, I chose to include new features such as console, genre, and publisher, along with critic_score. By including these new features, I'm trying to gain a better understanding of how a combination of game-specific attributes can affect the global sales.

First, I started by examining the distributions and potential relationships between these features and total_sales. For instance, I created a histogram of console and sales:



It's hard to see because there's a lot of consoles, but this shows that certain consoles, like the PSP, have a higher overall sales trend.

Now, I also decided to make a scatter plot of critic_score versus total_sales for each genre so I could see if certain genres benefit more from high critic scores.



As I can observe, genres like action, shooter, and action-adventure tend to have more sales, and higher critic scores.

Experiment 2 - Pre-processing step:

For this experiment, I handled missing values for the newly added features. Previously, I filled missing values in critic_score and total_sales with the median. For categorical features like console, genre, and publisher, I used **Label Encoding** to convert them into numerical values, so the regression model can use them:

```
Encode columns being used:

label_encoder = LabelEncoder()

exp_2df = game_df

for column in ['console', 'genre', 'publisher']:
    exp_2df[column] = label_encoder.fit_transform(exp_2df[column])

exp_2df
✓ 0.0s
```

	console	genre	publisher	developer	critic_score	total_sales	na_sales	jp_sales	pal_sales	other_sales	release_date	last_update
0	54	0	2445	Rockstar North	9.4	2032	6.37	0.99	9.85	3.12	2013-09-17	NaN
1	55	0	2445	Rockstar North	9.7	1939	6.06	0.60	9.71	3.02	2014-11-18	2018-01-03
2	53	0	2445	Rockstar North	9.6	16.15	8.41	0.47	5.49	1.78	2002-10-28	NaN
3	73	0	2445	Rockstar North	7.5	15.86	9.06	0.06	5.33	1.42	2013-09-17	NaN
4	55	15	101	Treyarch	8.1	15.09	6.18	0.41	6.05	2.44	2015-11-06	2018-01-14
...
64011	49	19	131	Arc System Works	7.5	0.12	NaN	NaN	NaN	NaN	2016-08-11	2019-01-28
64012	55	19	2023	Nippon Ichi Software	7.5	0.12	NaN	NaN	NaN	NaN	2020-07-30	2020-05-09
64013	45	19	2023	Nippon Ichi Software	7.5	0.12	NaN	NaN	NaN	NaN	2020-07-30	2020-05-09
64014	45	19	1349	Otomate	7.5	0.12	NaN	NaN	NaN	NaN	2019-02-28	2019-02-24
64015	55	19	3062	G.rev Ltd.	7.5	0.12	NaN	NaN	NaN	NaN	NaN	2023-09-29

64016 rows x 12 columns

Previously, there were also some more missing values, but still the features included do not have missing values:

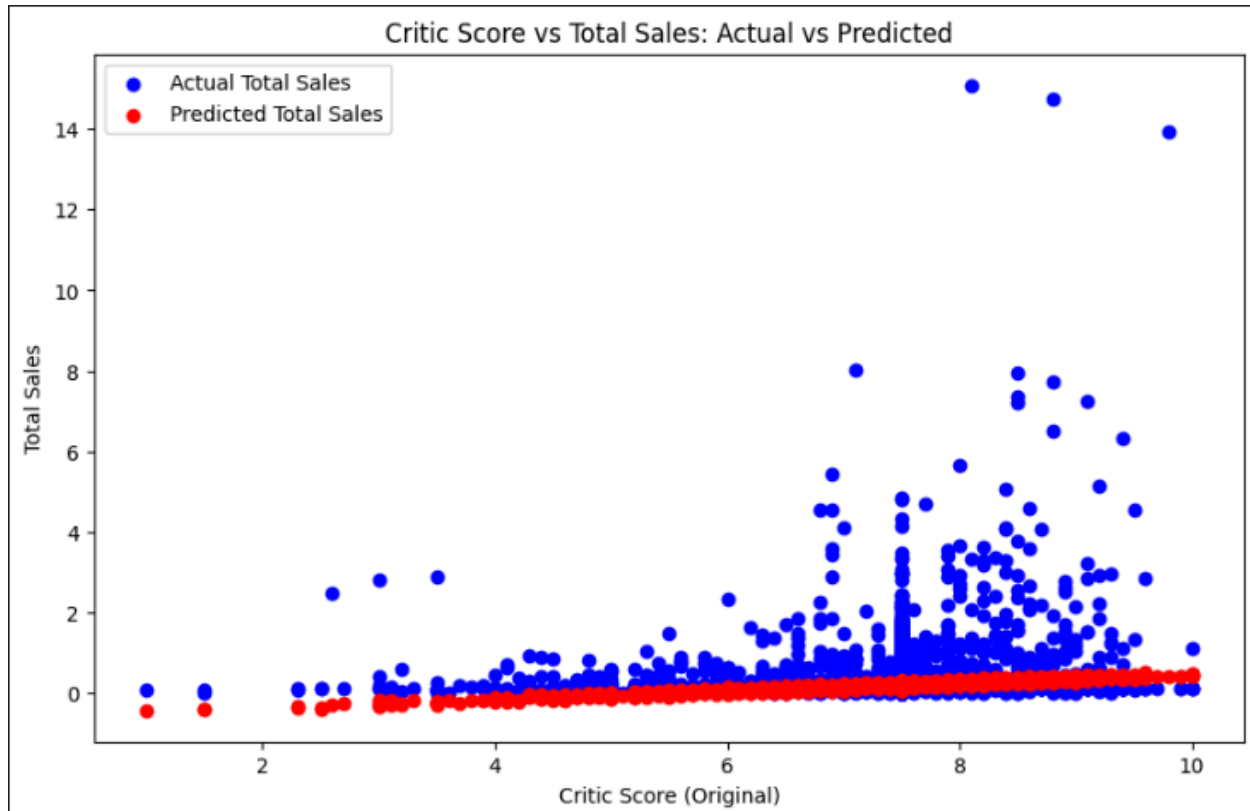
```
title           0
console         0
genre           0
publisher       0
developer       17
critic_score    57338
total_sales     45094
na_sales        51379
jp_sales        57290
pal_sales       51192
other_sales     48888
release_date    7051
last_update     46137
dtype: int64
```

The pre-processed data now includes:

- **Independent Variables:** critic_score, console, genre, publisher
- **Dependent Variable:** total_sales

Experiment 2 - Modeling:

Now, I created a new linear regression model using the expanded set of features. As before, I split the dataset into training and testing sets (80% training, 20% testing). Using Scikit-learn's LinearRegression, I trained the model on this new data to see how the added features impact the accuracy of predictions.



Upon first look, it looks almost exactly the same; however, it is not exactly the same as seen when we evaluate the model.

Experiment 2 - Evaluation:

Once again, I evaluated the model with Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared.

```
mse_1 = mean_squared_error(y_test1, y_pred1)
r2_1 = r2_score(y_test1, y_pred1)
rmse_1 = np.sqrt(mse_1)
✓ 0.0s

print(f"Mean Squared Error (MSE): {mse_1}")
print(f"Root Mean Squared Error (RMSE): {rmse_1}")
print(f"R-squared: {r2_1}")
✓ 0.0s

Mean Squared Error (MSE): 0.1817952218386695
Root Mean Squared Error (RMSE): 0.4263745088987726
R-squared: 0.013877274411377316
```

When evaluating the performance, my new “expanded” model compared the initial from experiment one with fewer features, I did observe some slight improvements. The **Mean Squared Error (MSE)** for the new model decreased from **0.18267** to **0.18180**, and the **Root Mean Squared Error (RMSE)** also showed a small reduction from **0.4274** to **0.4263**.

These changes suggest that the adding these new features: console, genre, and publisher, has slightly improved the accuracy of the predictions, though the effect is not very noticeable.

However, the R-squared value increased from **0.0091** to **0.0139**, showing that the new model is capturing more variance within the data, but still it is at a low level.

Overall, this tells me that while the added features contribute to some level of improvement, the overall effect is still not very beneficial. Also, it suggests that there may be other important factors that are still not captured in this model, and that a different approach might be needed, and for experiment 3 I will be adding all the features.

Experiment 3 - Pre-processing:

For this final experiment, I ensured all features were in numerical form and I removed all remaining missing values. Categorical variables (console, genre, publisher, developer) are now label-encoded. I dropped any columns that still had high percentages of missing data after the initial handling. I also converted the dates into just the year, so I can make the model more easily.

The updated feature set for Experiment 3 includes:

- **Independent Variables:** critic_score, console, genre, publisher, developer, release_date, na_sales, jp_sales, pal_sales, other_sales
- **Dependent Variable:** total_sales

Images of changes:

```
exp_3df = exp_2df.copy()
exp_3df.dropna(inplace=True)
exp_3df
```

✓ 0.0s

	console	genre	publisher	developer	critic_score	total_sales	na_sales	jp_sales	pal_sales	other_sales	release_date	last_update
1	55	0	2445	Rockstar North	9.7	19.39	6.06	0.60	9.71	3.02	2014-11-18	2018-01-03
4	55	15	101	Treyarch	8.1	15.09	6.18	0.41	6.05	2.44	2015-11-06	2018-01-14
7	55	1	2445	Rockstar Games	9.8	13.94	5.26	0.21	6.21	2.26	2018-10-26	2018-11-02
8	73	15	101	Treyarch	8.4	13.86	8.27	0.07	4.32	1.20	2012-11-13	2018-04-07
9	54	15	101	Treyarch	8.0	13.80	4.99	0.65	5.88	2.28	2012-11-13	2018-04-07
...
13130	59	13	1937	Kadokawa Games	7.5	0.05	0.01	0.03	0.00	0.00	2017-06-20	2018-09-17
13169	54	13	1937	Kadokawa Games	7.5	0.05	0.03	0.01	0.01	0.01	2014-09-30	2018-11-23
13198	53	18	1937	Vanillaware	7.7	0.05	0.01	0.03	0.01	0.00	2007-06-26	2019-01-10
13642	59	5	131	Examu Inc.	7.5	0.04	0.02	0.02	0.00	0.01	2014-09-23	2019-04-25
15044	2	0	1947	Delta Factory	4.5	0.02	0.01	0.00	0.01	0.00	2014-10-07	2019-01-24

690 rows x 12 columns

```
exp_3df.isnull().sum()
✓ 0.0s

console      0
genre        0
publisher    0
developer    0
critic_score 0
total_sales  0
na_sales     0
jp_sales     0
pal_sales    0
other_sales  0
release_date 0
last_update  0
dtype: int64
```

```
label_encoder = LabelEncoder()
exp_3df['developer'] = label_encoder.fit_transform(exp_3df['developer'])

exp_3df['release_date'] = exp_3df['release_date'].str[:4].astype(int)
exp_3df['last_update'] = exp_3df['last_update'].str[:4].astype(int)

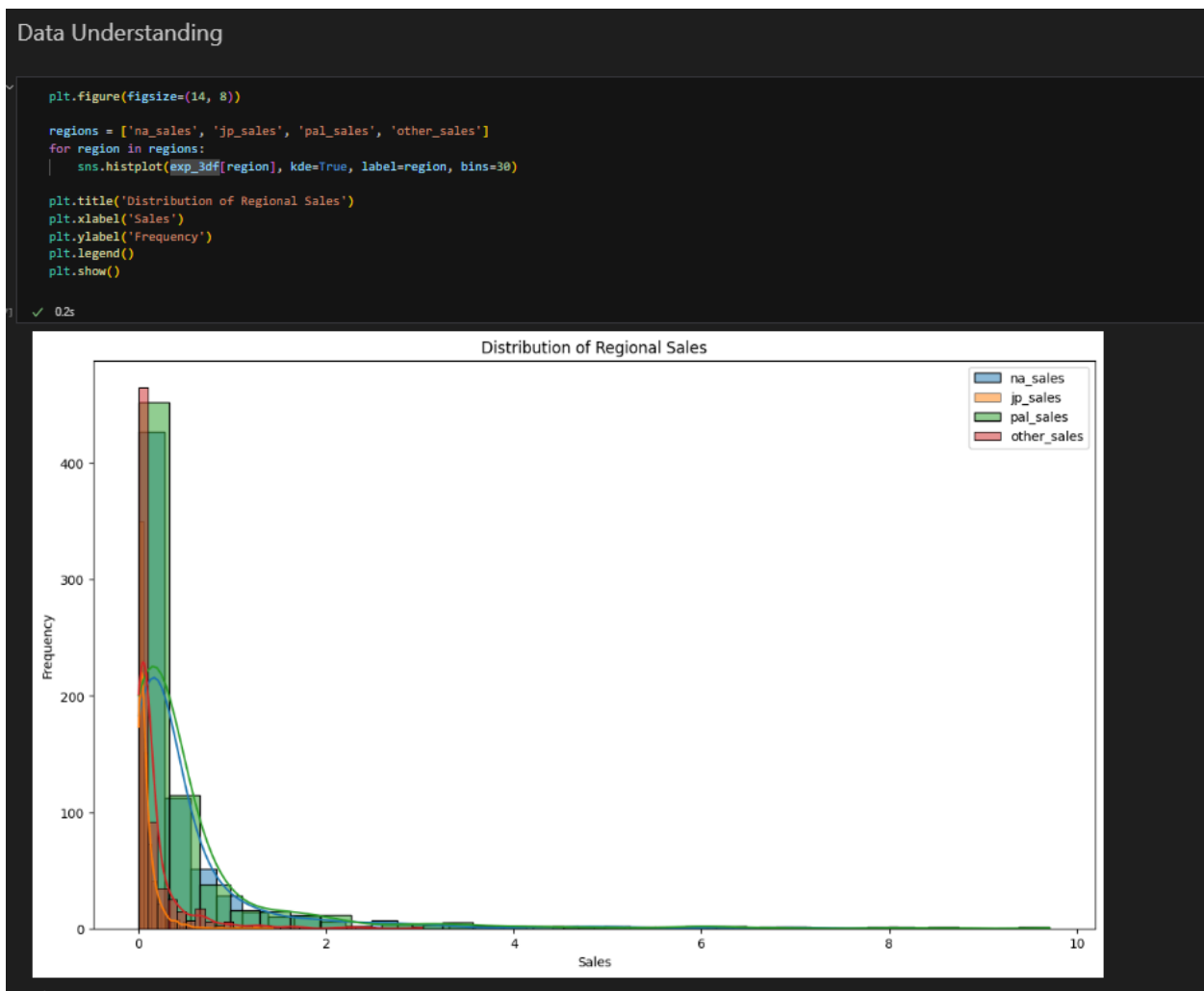
exp_3df
✓ 0.0s
```

	console	genre	publisher	developer	critic_score	total_sales	na_sales	jp_sales	pal_sales	other_sales	release_date	last_update
1	55	0	2445	176	9.7	19.39	6.06	0.60	9.71	3.02	2014	2018
4	55	15	101	217	8.1	15.09	6.18	0.41	6.05	2.44	2015	2018
7	55	1	2445	175	9.8	13.94	5.26	0.21	6.21	2.26	2018	2018
8	73	15	101	217	8.4	13.86	8.27	0.07	4.32	1.20	2012	2018
9	54	15	101	217	8.0	13.80	4.99	0.65	5.88	2.28	2012	2018
--	--	--	--	--	--	--	--	--	--	--	--	--
13130	59	13	1937	116	7.5	0.05	0.01	0.03	0.00	0.00	2017	2018
13169	54	13	1937	116	7.5	0.05	0.03	0.01	0.01	0.01	2014	2018
13198	53	18	1937	230	7.7	0.05	0.01	0.03	0.01	0.00	2007	2019
13642	59	5	131	74	7.5	0.04	0.02	0.02	0.00	0.01	2014	2019
15044	2	0	1947	57	4.5	0.02	0.01	0.00	0.01	0.00	2014	2019

Experiment 3 - Data Understanding:

In Experiment 3, I used the complete set of available features in the dataset to see if adding even more features will lead to improvements. The chosen features now include critic_score, console, genre, publisher, developer, release_date, and regional sales (na_sales, jp_sales, pal_sales, other_sales).

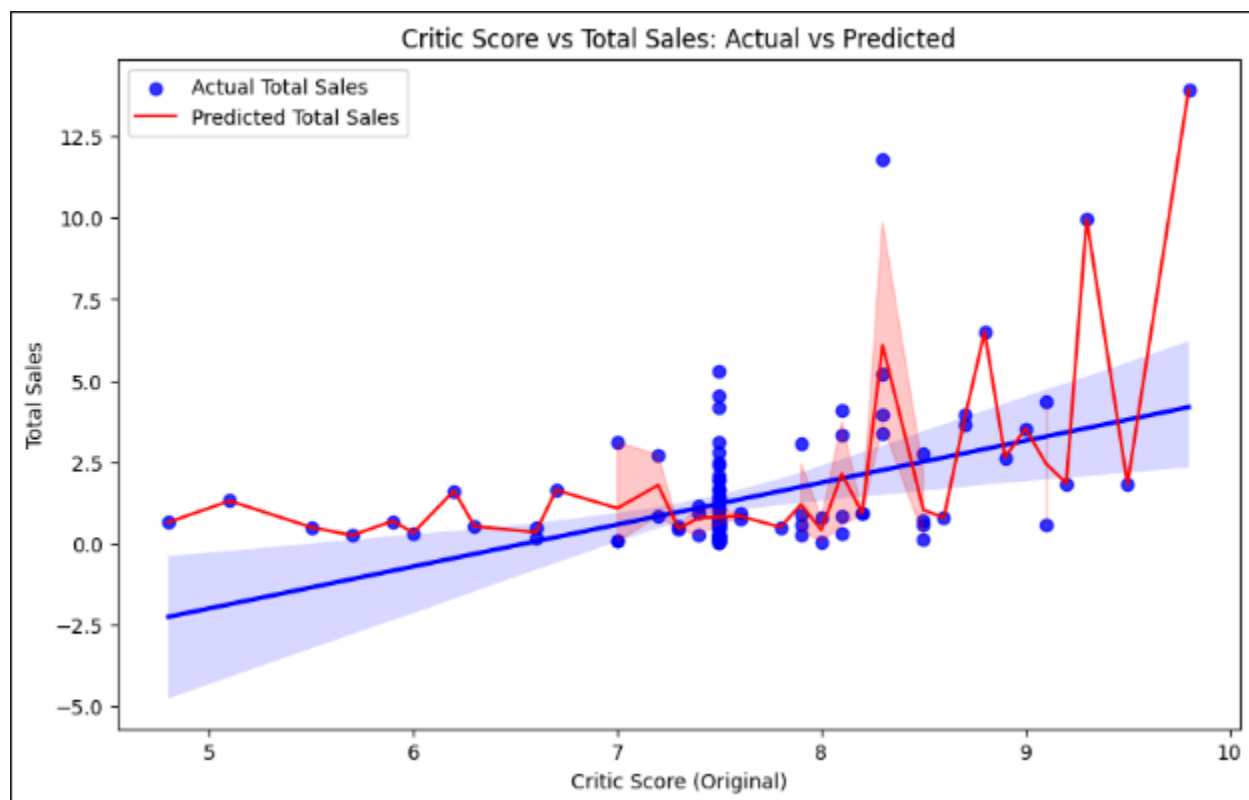
First, I made a model showing the distribution of regional sales across North America (na_sales), Japan (jp_sales), Europe and Africa (pal_sales), and other regions (other_sales) to better understand the variability and distribution patterns within the dataset.



As you can see in this visualization, it shows that all regions exhibit a right-skewed distribution, showing that most games have low sales, with only a few achieving high sales. It looks like PAL tends to have higher sales compared to other regions, suggesting maybe a more available market, while Japan and America also contribute significantly, but to a lesser extent. The "other" regions have the least contribution in terms of sales. This disparity suggests that maybe game publishers need to consider better ways to maximize sales.

Experiment 3 - Modeling:

Using this final complete feature set, I trained a new linear regression model. The data was again split into 80% training and 20% testing. This model uses all available features, so hopefully I see some better results:

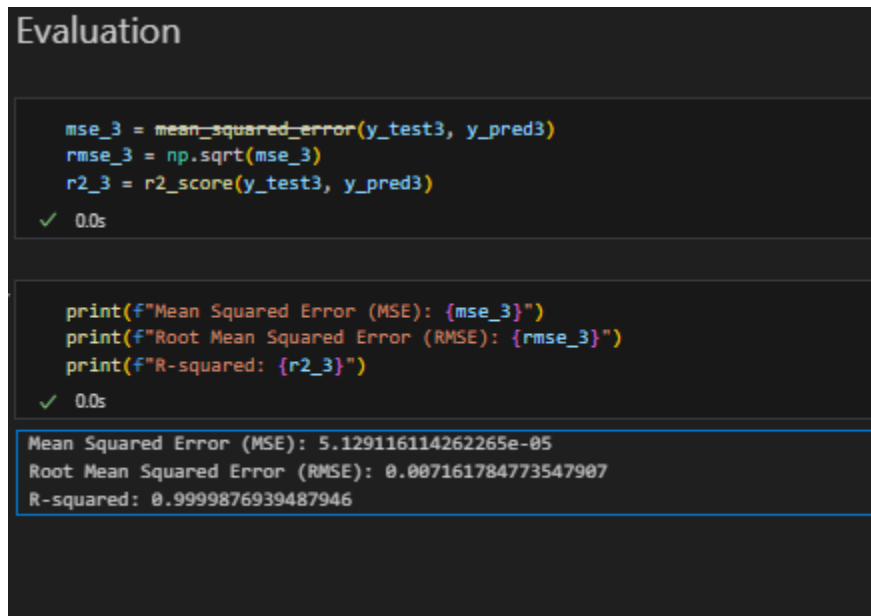


As we can see the graph shows the comparison between the **actual total sales** (blue dots) and the **predicted total sales** (red line) based on critic_score. The blue regression line with a shaded confidence interval indicates a slight positive correlation

between critic scores and total sales, suggesting that higher critic scores may generally lead to higher sales, but the trend doesn't seem that strong. The red line that shows the model's predictions fluctuates quite a bit, and this could mean that the model does not accurately capture the actual trend in the data. There are some noticeable differences between actual and predicted values, particularly at higher critic scores, which suggests that the model struggles to predict total sales accurately using only critic scores. Though, let's see what the evaluations provide below as the visual isn't the greatest at showing.

Experiment 3 - Evaluation:

The model was evaluated using MSE, RMSE, and R-squared as before.



```
Evaluation

mse_3 = mean_squared_error(y_test3, y_pred3)
rmse_3 = np.sqrt(mse_3)
r2_3 = r2_score(y_test3, y_pred3)
✓ 0.0s

print(f"Mean Squared Error (MSE): {mse_3}")
print(f"Root Mean Squared Error (RMSE): {rmse_3}")
print(f"R-squared: {r2_3}")
✓ 0.0s

Mean Squared Error (MSE): 5.129116114262265e-05
Root Mean Squared Error (RMSE): 0.007161784773547907
R-squared: 0.9999876939487946
```

After looking at the evaluation metrics, they show that model performed extremely well, maybe too well.

The **R-squared value of 0.9999**, shows that the model explains almost all the variance in the total sales data. The **Mean Squared Error (MSE)** and **Root Mean Squared Error (RMSE)** are also very close to zero, suggesting that the difference between the actual and predicted values is minimal.

Even though I'm glad the model brought back these "successful" results, this high R-squared value might tell me that **overfitting** is occurring especially since the visual output shows that the predicted values aren't really aligning well with the actual trend.

Impact Section:

First, since I was using a limited dataset, it can affect the model's performance, as the game sales data might have too many of a certain console or genre and cause the other data points to be unrepresented. With this, it can lead to bias in the predictions I made, and it could be bad for genres that are lesser-known, as well as development studios. These biases could unfairly affect predictions for the smaller or less conventional games, possibly giving an inaccurate showing of the potential sales that they will make.

Another issue that can occur is that game studios and publishers could over-rely on this sales prediction model. By relying too heavily on my model's predictions, it could lead companies to prioritize game features that are associated with higher predicted sales, rather than focusing on creativity or different/unique game designs. With this in mind, it probably would negatively impact the overall diversity of games being developed, as studios might choose safer options, like making a shooting game or action game, so they can maximize profits instead of pursuing fun and actually interesting game ideas.

There could be some cultural biases that might also be present, as the dataset I used may have more Western games, genres, and consoles, that lead to predictions that do not provide a good image for other markets. This could lead to a lack of visibility for games that come from different cultural backgrounds, and could limit success for those in other countries.

For ethical concerns, there could be some concerns that predictive models like mine, might influence which games are promoted, affecting what games the player chooses. Once again this could impact what genres or types of games become successful, supporting the idea that they prioritize profitability over creativity and innovation.

In conclusion, while the primary goal of my project is to understand what factors drive video game sales and to develop a predictive model, it's important to understand the impacts models like mine could have.

Conclusion:

Throughout the project, I learned through steadily increasing the feature selection and removing a lot of the missing_values near the end at experiment 3, can have a drastic effect on the performance of a regression model. In Experiment 1, using only critic_score as the feature resulted in a poor model fit, as shown by low R-squared values, high RMSE values, and the scatter plots showing not so accurate predictions. Experiment 2, which added some additional features such as console, genre, and publisher, showed some improvements, but the model still struggled to fully capture the complexity of total_sales. Finally, in Experiment 3, I included all available features like developer, regional sales data, and release_date, and the model returned a significantly higher R-squared value and lower error scores, showing that there's a much better fit with these features. However, seeing this extremely high R-squared value, and low RMSE value in Experiment 3, combined with some inaccuracies observed in the visual outputs, shows that there could be some overfitting happening.

The pre-processing steps, such as handling missing values and label encoding categorical features, were important for enabling the model to work with the data effectively. Also, I observed that incorporating additional features didn't really improve the model's performance, but by removing all the missing values I think that's what increased the success of the model. Also adding more features, increased the risk of overfitting, and it may have happened. Overall, I learned to pre-process more accurately, and it's very important to be careful of my evaluations, as it could be a result of overfitting.

References:

Data Pipeline Resource:

<https://www.beautiful.ai/player/-NoO5oPOTi1lShJwrDCs/2-The-Pipeline-Defining-the-Problem-and-Understanding-Data>

Linear Regression Resource:

<https://www.geeksforgeeks.org/ml-linear-regression/>

Visualizing Linear Regression Resource:

<https://python-graph-gallery.com/556-visualize-linear-regression/>

RMSE resource:

<https://c3.ai/glossary/data-science/root-mean-square-error-rmse/>