Statistics for DS Assignment 1, by Marius Ursu

Introduction

In order to help a game creator increase their sales, an online dataset covering a popular Board Games platform was analyzed.

This report will describe the data, the patterns present, and draw some conclusions for creating a successful board game.

Board games and the industry

Successful board games often share several common elements, although this may vary: engaging and balanced gameplay, clear and concise rules, high replay value, quality components, and appealing themes. Engaging gameplay typically involves strategic depth and player interaction, while balanced gameplay aims to ensure no single strategy dominates. Clear rules help prevent confusion, and high replay value can encourage players to return. Quality components may enhance the experience, and appealing themes can attract a diverse range of players.

The board game industry appears to have seen significant growth, particularly with the rise of crowdfunding platforms and a resurgence in tabletop gaming culture. This growth seems to have led to a wide variety of games catering to different tastes and complexity levels.

The target audience for board games is quite diverse, ranging from families and casual gamers to hobbyists and competitive players. Families and casual gamers might often seek games that are easy to learn and quick to play, while hobbyists and competitive players may prefer games with deeper strategic elements and longer playtimes. This diversity likely necessitates a broad range of games to meet varying preferences and play styles.

Performance in our dataset:

Factors which relate to a game's performance are:

Review `ratings` - mean ratings, not weighed by number of reviews

`Wishing`/`Wanting` - a measure of how many users want a particular game. Strongly correlated between each other, and with `Owned`

`Owned` - how many of the users own a game.

The next chapter will delve deeper into each of these features, and much more:

Chapter 1: Data Understanding

The top 20 most owned games are:

	id	name	owned	rank_ratings	average_ratings	users_rated
0	30549	Pandemic	168364	106	7.59	108975
2	13	Catan	167733	429	7.14	108024
1	822	Carcassonne	161299	190	7.42	108738
3	68448	7 Wonders	120466	73	7.74	89982
6	178900	Codenames	119753	101	7.6	74419
8	173346	7 Wonders Duel	111275	16	8.11	69472
4	36218	Dominion	106956	104	7.61	81561
5	9209	Ticket to Ride	105748	192	7.41	76171
7	167791	Terraforming Mars	101872	4	8.42	74216
17	129622	Love Letter	98395	293	7.23	59467
15	230802	Azul	95533	58	7.8	62802
11	148228	Splendor	92366	183	7.44	64698
14	14996	Ticket to Ride: Europe	92203	131	7.54	63264
16	70323	King of Tokyo	88090	345	7.17	61099
13	40692	Small World	86452	280	7.25	64559
12	169786	Scythe	86371	14	8.22	64569
19	266192	Wingspan	83920	23	8.1	56079
30	1927	Munchkin	78849	4796	5.89	43750
21	163412	Patchwork	78730	94	7.64	52190
9	31260	Agricola	78591	36	7.93	66093

Based on the correlation matrix, and feature selection regressions, the following features have demonstrated consistent ability to predict review ratings and ownership. They are candidates for IVs: `yearPublished` (year of publishing), `minPlayers` (minimum amoutn of players that

can play the game), `maxPlayers` (maximum players), `minAge` (rating of how old someone must be to play), `playtime` (playingtime in minutes)

`owned` - strong positive correlations exist between these numbers. They are collinear and will be accounted for in the studies. Games which have a higher number in one, are likely to have high numbers in all 4: `owned`, `trading`, `wanting`, `wishing`

And finally, there are a few measures relating to the rating of the game:

- `users_rated` - once again closely associated with a high number if one of the four above
- `average_rating` - the higher the rating, the more favorable a game is seen
- `bayes_average_rating` - average_rating weighed by the number of reviews (users_rated)

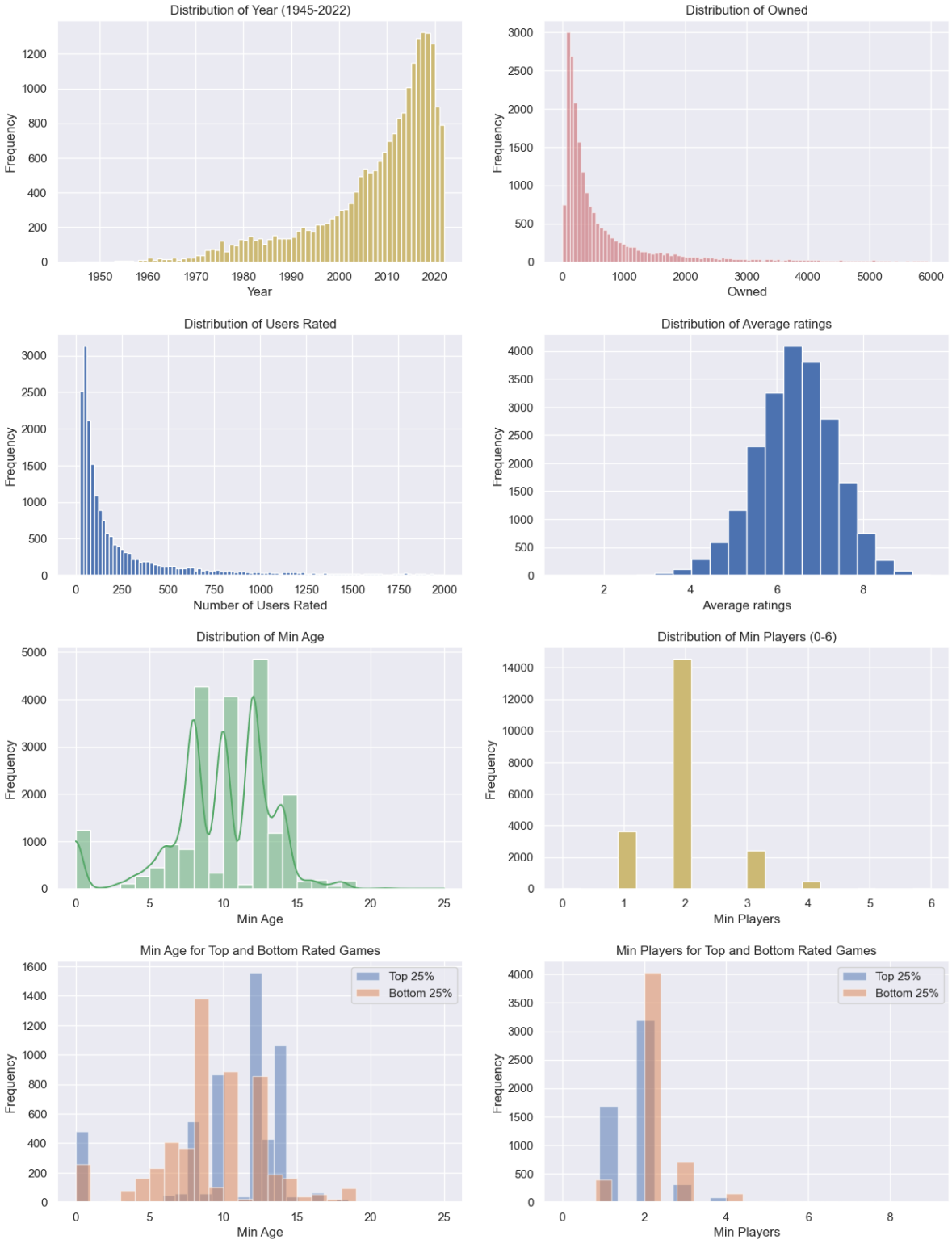

Figure 1: EDA Histogram

The plots tell us a few things:

- The distribution of Year shows that there are more recent games uploaded to the platform than older games. Background research shows that there is a delay of ~2 years between a game being released, and it making its way on the platform.
- The distribution of owned shows that most of the games reside at the lower end of the spectrum.
- How many users rated a particular game is heavily skewed to the left, meaning that there are many games which did not get rated.
- Average ratings resemble a normal distribution, with the mean situated between 6 and 7.
- Minimum age to play a game varies significantly, with 7-15 being the most common. On average, a higher minimum age is seen in highly rated games.
- Most games are multiplayer, requiring 2 or more players. At the same time, highly ranked games are more likely to be playable with one player.

Note: A filter for recent (yearPublished > 1945) games was employed: The goal of the project is discovering what makes money for the creator. That is done with a new game. Discovering a game that is not yet known is very unlikely. Simulating what made an old game successful might help, but we simply do not have enough features that describe the games - so that is out of the question.

Data preparation

For further analysis, scaling was performed in order to normalize the data, so that it is on the same scale. This way the coefficients can reflect what we want to learn out of them. Scales like Owned become linear on a log scale, as evidenced by scatterplots.

The outliers were not filtered in any way, as the author of this study does not have a good reason to remove data, until it can be pressumed to be erroneous. Upon running regressions, no Cook's distances surpassed 1, therefore we can conclude that there are no cases of anomalous data affecting the results.

A quick inspection of missing values showed that this is not the case with numerical columns, we therefore did not do any imputation or droping of entries.

Chapter 2: Regression and Analysis

As discussed in the introduction, success metrics consist of:

- How well rated a game is, as measured by `average_rating`. Note: `bayes_average` is derived using the `number of ratings`, which in turn is strongly associated with `owned`. This would provide limited analytical insight.

- How popular a game is, as measured by `owned` - which in turn is associated strongly with `wishing`, `wanting`, and a few other metrics.

Choosing independent variables for the DV: `average_rating`

Through a run of Lasso regression, which is recognized more for feature selection than it is for regularization, it was discovered that the following variables have a strong relation with the DV of `average_rating`.

Note : When predictors are highly correlated, Lasso tends to arbitrarily select one of them while setting the coefficients of the others to zero. This can lead to instability in the selected variables and make the interpretation of coefficients more challenging. This is why this analysis was combined with the correlation matrix, and an explainable variable was taken forward.

- `minplayers` - with a negative coefficient, meaning that games with a lower barrier to play get a higher rating.
- `minage` - with a positive coefficient. Games with a higher minimum age receive a more positive rating.

To run lasso regression for the DV of `average_rating`, it was important to filter the games based on at least 250 ratings. This helped us eliminate the noise in the data, which the model could not explain. For the purpose of this analysis: consider that if you make it past 250 ratings, these guidelines start to apply.

Findings

Linear regression in python is done using the control variables, which allows us to look at the coefficients and draw conclusions. To complete the analysis, we conduct one more run without including the control variables, to see their absence's effect on model performance.

The models predicted `rating_average` using the following variables:

1. **Full Model**: Includes `owned`, `minplayers`, `minage`, and `yearpublished`.
2. **Reduced Model**: Excludes `yearpublished`, includes `owned`, `minplayers`, and `minage`.
3. **Further Reduced Model**: Excludes both `yearpublished` and `owned`, includes `minplayers` and `minage`.

Here is the table with the intercept column removed:

Model	R^2	Adj. R^2	F-statistic	p-value	owned	min-players	min-age	year-published
Full Model	0.284	0.284	562.1	<0.001	0.1522	-0.1222	0.1636	0.2560
Reduced Model	0.170	0.170	387.4	<0.001	0.1562	-0.1541	0.1951	-

Model	R^2	Adj. R^2	F-statistic	p-value	owned	min-players	min-age	year-published
Further Reduced Model	0.127	0.126	410.5	<0.001	-	-0.1575	0.2043	-

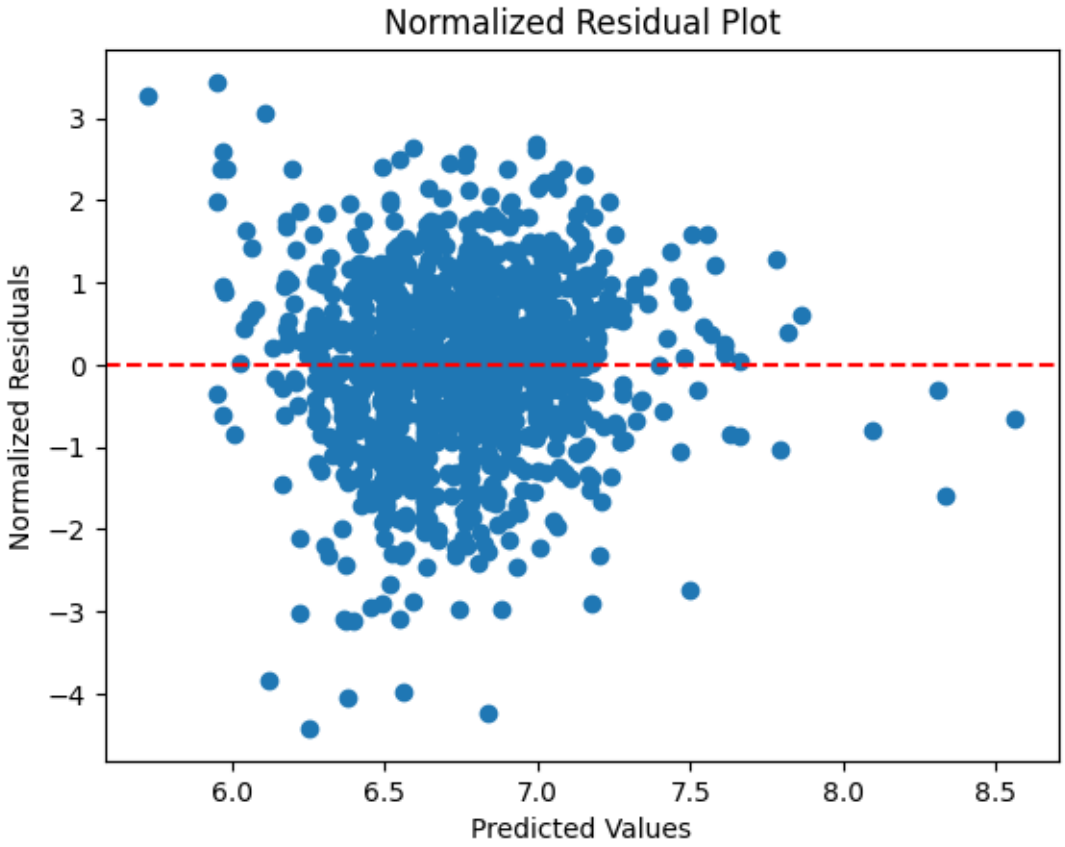

Figure 2: Normalized residuals against the Predicted

R^2 and Adjusted R^2

- **Further Reduced Model ($R^2 = 0.127$)**: Indicates that 12.7% of the variance in `rating_average` is explained by the model, excluding both `yearpublished` and `owned`.

The substantial drop in R^2 from the full model to the reduced models demonstrates the significant contribution of `yearpublished` in explaining the variance in `rating_average`.

This is in line with background research which described a positive relationship between more recent games and higher ratings.

F-statistic and Model Significance

- All models are statistically significant (p-value < 0.05), indicating that at least one of the predictors in each model has a significant relationship with the dependent variable.
- The full model's higher F-statistic (562.1) compared to the reduced models (387.4 and 410.5) suggests a better overall fit when `yearpublished` is included.

Coefficients

- **Intercept (const)**: The intercept remains at 6.7220 across all models, which is the expected `rating_average` when all predictors are at zero. The F-statistic tells us that the model performs better than simply using the mean.
- **minplayers**:
  - Full Model: Coefficient is -0.1222, indicating a negative relationship with `rating_average`.
  - Reduced Model: Coefficient becomes more negative at -0.1541.
  - Further Reduced Model: Coefficient is -0.1575.
- **minage**:
  - Full Model: Coefficient is 0.1636, indicating a positive relationship with `rating_average`.
  - Reduced Model: Coefficient increases to 0.1951.
  - Further Reduced Model: Coefficient further increases to 0.2043.

Discussion

The significance of all coefficients (p-values < 0.05) across all models indicates that each variable contributes meaningfully to the model. The findings emphasize the importance of controlling for relevant variables, such as `yearpublished`, to better understand the relationships between predictors and the dependent variable.

In our regression analysis, `yearpublished` and `owned` are important control variables that help us better understand how factors like `minplayers` and `minage` relate to game ratings. `Yearpublished` acts as a way to consider how time might affect ratings, capturing any changes in the board gaming industry over the years. Background research, and our own plots suggest that an increase in year coincides with an increase in rating. Similarly, `owned` gives us a sense of a game's popularity among players.

Including these control variables helps us address any factors that could bias our results. By controlling for `yearpublished` and `owned`, we can focus on the specific impact of `minplayers` and `minage` on ratings with more confidence. This approach strengthens the reliability of our findings and allows us to draw clearer conclusions about what influences board game ratings.

We therefore conclude that all 4 IV's help predict ratings. `year` and `owned` were used for controling, as they are not cannot be directly influenced by the creator of the new game. Furthermore, `minplayers` having a negative coefficient, means that games with a lower barrier to play get a higher rating. Based on the 1 IV to 1 DV regession chart, this relationship breaks down after a minimum of 5 players. `minage` - with a positive coefficient. Games with a higher minimum age receive a more positive rating. The website audience is primarily adults, which explains this effect.

Analysis 2: Ridge Regression for the DV: `owned`

When dealing with multicollinear features, Lasso (L1 regularization) tends to arbitrarily select one of the correlated features while setting the coefficients of the others to zero. On the other hand, Ridge regression (L2 regularization) tends to shrink the coefficients of correlated features towards each other without necessarily setting any to exactly zero. Because our goal is to retain multicollinear features in the model to observe their impact while still penalizing their coefficients to avoid overfitting, Ridge regression may be more suitable. By applying Ridge regularization, we can control the extent to which the coefficients are shrunk towards each other, allowing us to observe the relative importance of the correlated features.

Therefore, the second analysis we present is done using Ridge regression, which had its parameters lowered so that it mantains collinearity. We were interested to see which classes of game mechanic had a positive effect on how `owned` the game was.

There was no need to filter based on popularity, and doing so did not improve the amount of residuals explained by the model. By looking at the coefficients of the Lasso regression, as well as accounting for collinearity through the correlation matrix, the IVs with predictive power were found to be:

`minplayers`, `maxplayers`, `playingtime`, `minplaytime`, `maxplaytime`, `minage`, `Mechanic_` (185 columns)

What is new here is that we have looked at game mechanics: based on background research, we wanted to see how the categorical column influences how owned a game is. Therefore, the column was exploded, encoded, and used in the analysis.

Once again, due to how statistical models run in Python, we have to run two models, one with the control variables, and one without:

Full model

Including the control variables: `rating_average`, `yearPublished`

- R-squared: 0.102
- F-statistic: 2.44

- p-value: <0.001

Coefficients: `rating_average`: 402.75; `yearpublished`: -12.69; `minplayers`: 116.73; `maxplayers`: 36.55; `playingtime`: -46.13; `minplaytime`: 79.60; `maxplaytime`: -46.13; `minage`: 28.21

Reduced Model:

Reduced model, eliminating `yearPublished` and `rating_average` which acted as **control variables**. These are not within the creator's direct control. Based on previous analysis, we can conclude that it is a good time to publish, and as described before, positive ratings go hand in hand with popularity.

- R-squared: 0.098
- F-statistic: 2.36
- p-value: <0.001

Coefficients: `minplayers`: 86.03; `maxplayers`: 32.03; `playingtime`: -8.20; `minplaytime`: 17.86; `maxplaytime`: -8.20; `minage`: 50.27 and `Mechanic_` which did not change meaningfully between models, to be discussed shortly.

Comparison

R-squared: The R-squared value decreased slightly from approximately 10.2% in the full model to about 9.8% in the reduced model. This suggests that the reduced model explains slightly less variance in the dependent variable compared to the full model.

Despite the reduction in model complexity, the reduced model still maintains statistically significant coefficients and F-statistic, suggesting its utility in explaining the variance in the dependent variable.

The game mechanic features with the top 5 largest coefficients were: `Mechanic_Delayed Purchase` `Mechanic_Map Addition` `Mechanic_Random Production` `Mechanic_Hidden Roles` `Mechanic_End Game Bonuses`

These coefficients are dominating through their size and sheer number. There is a reglularization issue here which is better described in the limitations chapter.

Takeaways:

- `average_rating` - which makes sense: there is a reason why games become popular, and it has something to do with what people view them.
- `yearpublished` - negative coefficient, meaning that older games tend to be more owned - they had more time to become recognizable and be bought.
- `minplayers` and `maxplayers` - both with positive coefficients, meaning that the more multi-player a game is, the more owned it is. This is different than the results obtained previously for ratings, which had a negative coefficient until `minplayers` reached 5.
- `minplayingtime` - the more hours a game is played, the more it is owned.
- `minage` - with a strong positive coefficient: games that cater to a more adult audience are more likely to be owned by this community.
- `game mechanic` - the results suggest that there are themes in the types of games that make it to the top. A better understanding of these mechanics, and which genre they are part of would be essential to further interpret this information. This is further discussed in limitations too.

Assumptions and Generalizeability:

- Anomalous data affecting the results: No Cook's distance of over 1 were detected
- Normality of residuals: a visual inspection of the histogram, as well as summary statistics confirm that the residuals follow a normal curve
- Homoscedasticity - scatterplot in Figure 2 confirmed that there is no cone or discernible pattern. The residuals have a consistent variance against the predicted

We rely on the CLT to aid the generalizeability of these findings. The platform is of course not a random sample of the general population, which is why we conclude that the results found here apply to the platform only, and the suggestions should help in performance on the platform, not necessarily in the real world.

Based on contrasting the train vs test performance metrics, there were no signs of overfitting (test performance metrics were slightly less than train, as expected). This means that the model generalizes well, despite the fact that a large share of the variation in the DV remains unexplained.

This suggests a need for followup studies, looking at more specific types and segments of games. This would open the door for interaction effects, non-linear relationships, and even running tests with specific hypotheses to confirm certain limited findings.

Chapter 3: Recommendations and limitations

Disclaimer

Due to the high number of unexplained variance in the predicted variables, these suggestions must be considered carefully.

This analysis is not a substitute for conducting surveys, product-market fit studies, user-tests and other means of research. This ensures that feedback is available, and that it is timely.

The study author is not experienced enough to interpret many findings with reasonable confidence, and is not a Board Games Geek.

Findings

Multiplayer games: Games ames designed for more players are more likely to be owned. This suggests that board game enthusiasts prefer games that can offer more social interaction and can be enjoyed in a group setting. This finding contrasts with previous results for ratings, where games with a higher minimum player count had a negative coefficient until minplayers reached 5. Games which are accessible to smaller groups or can be played by fewer players could be preferred and rated more positively. Such games might be seen as more versatile and easier to set up, making them more appealing to a broader audience.

Game Duration: The positive coefficient implies that games with longer playing times are more likely to be owned. This suggests that the community values games that offer a more extended and possibly more immersive experience. Longer games might be perceived as providing better value or deeper gameplay, appealing to enthusiasts who enjoy investing time in complex, strategic games.

Age Suitability: The strong positive coefficient indicates that games designed for an older audience are more likely to be owned. This suggests that the board gaming community is primarily composed of adults who prefer games tailored to mature players. Games with higher age recommendations might include more complex rules, strategic depth, and mature themes, which appeal to adult gamers.

Ownership: A higher number of ownerships is correlated with higher ratings. This suggests a positive feedback loop: games that are liked and rated highly are more likely to be owned by many people. Conversely, the fact that many people own a game can be an indicator of its quality and popularity, leading to higher ratings.

Good Timing: A positive relationship between more recent games and higher ratings is consistent with background research. This trend might be due to improvements in game design, production quality, and innovative mechanics in newer games. It also reflects the tendency of players to rate recent purchases highly, perhaps influenced by the novelty and excitement of new games.

Recommendations

The creator of future board games is advised to strategically consider the minimum amount of players that can play. Successful games leverge multiplayer dynamics, but are also playable with a small group of people.

Catering to people with purchasing power, and those that can promote through word of mouth and other means is supported by our results (`minage` correlating positively with higher ratings and higher ownership).

Overall, these findings suggest that the board gaming community values games that support larger groups, offer longer playing times, and cater to an adult audience. This information can be useful for game designers and marketers looking to create or promote games that align with these preferences.

It is therefore important to leverage fans and clubs in the rollout and marketing campaigns, further described next:

Building on other studies

Leveraging the hype effect is important. This suggests a strong marketing campaign that builds on top of organic momentum in the community.

As stated by the dataset author in the original blog article: "Games from 2020 where rated with 7.8 in 2020, but two years later that has dropped to 7.2 and now games from 2022 are rated with 7.8! When games are about 5 years old the 'hype' effect is more or less gone and games reach a stable score (around 2014 there is no difference anymore)."

A different version of this dataset includes information about the complexity of the game. This analysis suggests that more complex games have a higher rating: "there is a relation between the complexity of the game and the score. More complex games get higher scores, it's almost a 1 point difference between a game with weight 1 and 5!".

The same analysis describes how "less active users give higher ratings" – which suggests that there are potential opportunities in activating those who are not active, and indeed bringing new people to the platform through nudging.

Limitations

The study author is not experienced enough to interpret many findings, and is not a Board Games Geek. Although some recent experiences, discussions, reading of articles and checking the platform that provided the data helped, a future iteration would include interviews with players, industry analysts in order to interpret these highlights and come up with market segmets which are worth catering to.

The data described in background research proves that at least two other dimensions would be key for understanding the dynamics of what makes a game successful:

- Detailed information about the game: how complex/challenging it is. This proved to have the highest predictive power in an analysis done by the dataset publisher.
- Who the users are which make or break its success, this information is again present in the larger dataset, and it allows for insights like "higher ratings are given by infrequent reviewers". This would allow the game creator to cater to specific audiences.

Other characteristics which could be used highlight patterns in games are: Type (e.g. Strategy), Category (e.g. Space exploration, Economic), Family, and even Publisher.

At this time, the study treats all users as the same, when in fact they can be ranging from families and casual gamers to hobbyists and competitive players. The nature of the data being collected introduces a sample bias. People who do it casually are unlikely to participate in online communities. Reviews in themselves bring up other kinds of biases (tend to be skewed toward very positive and very negative - the vocal people dominate)

When attempting to use game mechanics to detect patterns in the popular games, regularization techniques did not sufficiently penalize the `Mechanics_` categorical features. This is evidenced by the eliminating of control variables `year` and `average_rating` did very little for lowering the model perforamnce. The categorical features dominated the analysis. This requires a followup approach to hyper parameter tuning such that the collinear features are kept, to mantain explainability, but penalized so as to avoid overfitting and dominating the other coefficients through their sheer number.

Other techniques which work very well with a large number of categorical features are Decision Trees - which could further help in choosing features, or doing a new regression.

Although the models generalize well (significant results, without overfitting), they do not explain a large share of the variation in the predicted. This calls for more specific hypotheses, covering particular groups in the audience (requiring the larger dataset).

Applying a narrower filter of what kinds of games or success type we are looking at would drastically increase how much variance the model would explain. An example could be: focusing attention more on the games that did well within a few months of publishing: what do they have in common ? Other dynamics come into play when a game is years or even decades or hundreds of years old - with a loyal following. Another window would be focusing on particular types and families of games that are on the rise. This must be done carefully, as it eliminates much of the data. It builds into the study certain assumptions. Doing so would have to be approved by the main stakehodler / game creator.

Appendix:

Regression plots and correlation matrices can be seen here:

https://github.com/ursumarius/linear-regression-board-games/blob/main/eda-regression.ipynb

Lasso regression for Linear model feature selection

https://github.com/ursumarius/linear-regression-board-games/blob/main/avg_r_lasso-regression.ipynb

libraries used:

```
#import libraries
import numpy as np
import pandas as pd
import pyarrow

from collections import Counter
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
import seaborn as sns
sns.set_theme()

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from scipy.stats import f as f_statistic
from scipy.stats import pearsonr
import statsmodels.api as sm
```

Data ingestion, cleaning and preparation

```
rg = pd.read_csv("ratings.csv")
rg.head()

# Displaying hidden columns
pd.set_option('display.max_columns', None)

# Data ingestion
dt= pd.read_csv("./details.csv")
dt.head()


#drop year from rg as it correlates with year in details 1:1
rg.drop('year', axis=1, inplace=True)

#rename columns from ratings for clarity
rg.rename(columns={'num': 'num_r', 'rank': 'rank_ratings', 'average': 'average_ratings', 'bay

#join the two tables
gd = rg.merge(dt, on='id')

# lets see if the nan values are present in the dataset
gd.isnull().sum()

# Drop unnecessary columns
columns_to_drop = ['url', 'thumbnail', 'description', 'boardgamepublisher', 'boardgamedesigne
cleaned_data = combined_data.drop(columns=columns_to_drop)

# Convert categorical data to numeric format using one-hot encoding
# cleaned_data = pd.get_dummies(cleaned_data, columns=['boardgamecategory', 'boardgamemechani
# cleaned_data = cleaned_data.drop(['boardgamecategory', 'boardgamemechanic', 'boardgamefamil

# Check for duplicates and drop them
cleaned_data = cleaned_data.drop_duplicates()


cat = cleaned_data.select_dtypes(include=['object'])
#drop categorical because encoding and scaling is troublesome for regression.
cleaned_data_num = cleaned_data.drop(columns=cat.columns)


# No need to handle Nan as they fall in the categorical columns
```

```
# year does have outliers, a simple search on the internet could detect few. We assume the da

#we will focus our analysis on games published after 1945.
gdf = gd[gd['yearpublished'] >= 1945]

#lets select the categorical coluns to make some count plots
gdc = gdf.select_dtypes(include=['object'])
gdc

#drop categorical because encoding and scaling is troublesome for regression.
# publisher might be interesting as a predictor.
gdn = gdf.drop(columns=gdc.columns)
```

EDA

```
fig, axs = plt.subplots(4, 2, figsize=(15, 20))


# Plot the distribution of yearpublished
axs[0, 0].hist(gdn['yearpublished'], bins=77, color='y', range=(1945, 2022))
axs[0, 0].set_xlabel('Year')
axs[0, 0].set_ylabel('Frequency')
axs[0, 0].set_title('Distribution of Year (1945-2022)')

# Distribution of owned
axs[0, 1].hist(gdn['owned'], bins=100, range=(0, 6000), color='r', alpha=0.5, label='Owned')
axs[0, 1].set_xlabel('Owned')
axs[0, 1].set_ylabel('Frequency')
axs[0, 1].set_title('Distribution of Owned')

# Plotting the distribution of users_rated
axs[1, 0].hist(gdn['users_rated'], bins=100, range=(0, 2000))
axs[1, 0].set_xlabel('Number of Users Rated')
axs[1, 0].set_ylabel('Frequency')
axs[1, 0].set_title('Distribution of Users Rated')

# Plot the distribution of average ratings
axs[1, 1].hist(gdn['average_ratings'], bins=20)
axs[1, 1].set_xlabel('Average ratings')
axs[1, 1].set_ylabel('Frequency')
```

```
axs[1, 1].set_title('Distribution of Average ratings')

# Plot the distribution of minAge
sns.histplot(gdn['minage'], kde=True, bins=25, color='g', ax=axs[2, 0])
axs[2, 0].set_xlabel('Min Age')
axs[2, 0].set_ylabel('Frequency')
axs[2, 0].set_title('Distribution of Min Age')

# Plot the distribution of maxPlayers
axs[2, 1].hist(gdn['minplayers'], bins=20, color='y', range=(0, 6))
axs[2, 1].set_xlabel('Min Players')
axs[2, 1].set_ylabel('Frequency')
axs[2, 1].set_title('Distribution of Min Players (0-6)')


# Calculate the quantile values
top_quantile = gdn['average_ratings'].quantile(0.75)
bottom_quantile = gdn['average_ratings'].quantile(0.25)

# Filter for the top 25% most highly rated games
top_group = gdn[gdn['average_ratings'] >= top_quantile]

# Filter for the lowest 25% rated games
bottom_group = gdn[gdn['average_ratings'] <= bottom_quantile]

# Plot the distribution of minage for the top group
axs[3, 0].hist(top_group['minage'], bins=25, alpha=0.5, label='Top 25%')
# Plot the distribution of minage for the bottom group
axs[3, 0].hist(bottom_group['minage'], bins=25, alpha=0.5, label='Bottom 25%')
axs[3, 0].set_xlabel('Min Age')
axs[3, 0].set_ylabel('Frequency')
axs[3, 0].set_title('Min Age for Top and Bottom Rated Games')
axs[3, 0].legend()

# Plot the distribution of minplayers for the top group
axs[3, 1].hist(top_group['minplayers'], bins=20, alpha=0.5, label='Top 25%')
# Plot the distribution of minplayers for the bottom group
axs[3, 1].hist(bottom_group['minplayers'], bins=20, alpha=0.5, label='Bottom 25%')

axs[3, 1].set_xlabel('Min Players')
axs[3, 1].set_ylabel('Frequency')
axs[3, 1].set_title('Min Players for Top and Bottom Rated Games')
```

```
axs[3, 1].legend()

# Decrease plot height by 30%
plt.rcParams['figure.figsize'] = (plt.rcParams['figure.figsize'][0], plt.rcParams['figure.fig

# Increase spacing between subplots vertically
plt.subplots_adjust(hspace=0.3)

# Your code for creating and plotting the subplots goes here
plt.show()
```

Linear model predicting Ratings

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from tabulate import tabulate

# Prepare data for modeling
X_full = popular_cleaned_data_num[['owned', 'minplayers', 'minage', 'yearpublished']]
X_reduced = popular_cleaned_data_num[['minplayers', 'minage']]
y = popular_cleaned_data_num['rating_average']

# Split the data
X_train_full, X_test_full, y_train, y_test = train_test_split(X_full, y, test_size=0.2, rando
X_train_reduced, X_test_reduced, _, _ = train_test_split(X_reduced, y, test_size=0.2, random_

# Standardize the data
scaler_full = StandardScaler()
X_train_full_scaled = scaler_full.fit_transform(X_train_full)
X_test_full_scaled = scaler_full.transform(X_test_full)

scaler_reduced = StandardScaler()
X_train_reduced_scaled = scaler_reduced.fit_transform(X_train_reduced)
X_test_reduced_scaled = scaler_reduced.transform(X_test_reduced)

# Add a constant term for the intercept in statsmodels
X_train_full_scaled = sm.add_constant(X_train_full_scaled)
```

```
X_test_full_scaled = sm.add_constant(X_test_full_scaled)
X_train_reduced_scaled = sm.add_constant(X_train_reduced_scaled)
X_test_reduced_scaled = sm.add_constant(X_test_reduced_scaled)

# Fit the models using statsmodels
model_full = sm.OLS(y_train, X_train_full_scaled).fit()
model_reduced = sm.OLS(y_train, X_train_reduced_scaled).fit()

# Get the model summaries
summary_full = model_full.summary()
summary_reduced = model_reduced.summary()
print("Full Model Summary:")
print(summary_full)
print("\nModel excluding Owned, YearPublished - Summary:")
print(summary_reduced)

# Predict on the test data
y_pred_test_full = model_full.predict(X_test_full_scaled)
y_pred_test_reduced = model_reduced.predict(X_test_reduced_scaled)



# Get the summary table
summary_full = model_full.summary()

# Extract the desired statistics
dep_variable = summary_full.tables[0].data[0][1]
r_squared = float(summary_full.tables[0].data[0][3])
adj_r_squared = float(summary_full.tables[0].data[1][3])
f_statistic = float(summary_full.tables[0].data[2][3])
df_residuals = (summary_full.tables[0].data[6][1])
df_model_only = (summary_full.tables[0].data[7][1])

# Create a new table to display the statistics
table_data = [
    ['Dep. Variable:', dep_variable],
    ['R-squared:', r_squared],
    ['Adj. R-squared:', adj_r_squared],
    ['F-statistic:', f_statistic],
    ['DF Residuals:', df_residuals],
    ['DF Model only:', df_model_only]
]
```

```
table_headers = ['Statistic', 'Value']

# Display the table
print(tabulate(table_data, headers=table_headers))


# Get the summary table
summary_reduced = model_reduced.summary()

# Extract the desired statistics
dep_variable = summary_reduced.tables[0].data[0][1]
r_squared = float(summary_reduced.tables[0].data[0][3])
adj_r_squared = float(summary_reduced.tables[0].data[1][3])
f_statistic = float(summary_reduced.tables[0].data[2][3])
df_residuals = (summary_reduced.tables[0].data[6][1])
df_model_only = (summary_reduced.tables[0].data[7][1])

# Create a new table to display the statistics
table_data = [
    ['Dep. Variable:', dep_variable],
    ['R-squared:', r_squared],
    ['Adj. R-squared:', adj_r_squared],
    ['F-statistic:', f_statistic],
    ['DF Residuals:', df_residuals],
    ['DF Model only:', df_model_only]
]

table_headers = ['Statistic', 'Value']

# Display the table
print(tabulate(table_data, headers=table_headers))


# OLS Full Model Summary
ols_full_summary = model_full.summary()
ols_full_table = ols_full_summary.tables[1]
ols_full_headers = ols_full_table.pop(0)
ols_full_table_markdown = tabulate(ols_full_table, headers=ols_full_headers, tablefmt="pipe")

# OLS Reduced Model Summary
ols_reduced_summary = model_reduced.summary()
ols_reduced_table = ols_reduced_summary.tables[1]
```

```
ols_reduced_headers = ols_reduced_table.pop(0)
ols_reduced_table_markdown = tabulate(ols_reduced_table, headers=ols_reduced_headers, tablefr

print("OLS Full Model Summary:")
print(ols_full_table_markdown)
print("\nOLS Reduced Model Summary:")
print(ols_reduced_table_markdown)


# Calculate the residuals
residuals = y_test - y_pred_test

# Normalize the residuals
normalized_residuals = residuals / np.std(residuals)

# Plot the normalized residuals against the predicted values
plt.scatter(y_pred_test, normalized_residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Normalized Residuals')
plt.title('Normalized Residual Plot')
plt.show()
```

Ridge model for predicting Owned

```
# Prepare data for modeling
X_full = cleaned_data_encoded.drop(columns=[
    'id',  # Not a feature
    'num_rating',  # Not a feature
    'num_details',  # Not a feature
    'owned',  # Dependent variable
    'rating_bayes_average',  # Derived value strongly correlated with DV
    'wishing',  # Strongly correlated with DV
    'wanting',  # Strongly correlated with DV
    'trading',  # Strongly correlated with DV
    'rating_rank',  # Strongly correlated with another independent variable
    'rating_users_rated',  # Strongly correlated with another independent variable
    'year',  # Strongly correlated with yearpublished
])
```

```
X_reduced = cleaned_data_encoded.drop(columns=[
    'id',  # Not a feature
    'num_rating',  # Not a feature
    'num_details',  # Not a feature
    'owned',  # Dependent variable
    'rating_bayes_average',  # Derived value strongly correlated with DV
    'wishing',  # Strongly correlated with DV
    'wanting',  # Strongly correlated with DV
    'trading',  # Strongly correlated with DV
    'rating_rank',  # Strongly correlated with another independent variable
    'rating_users_rated',  # Strongly correlated with another independent variable
    'year',  # Strongly correlated with yearpublished
    'yearpublished',  # Want to see the effect of controlling for it
    'rating_average',  # Want to see the effect of controlling for it
])

y = cleaned_data_encoded['owned']


# Explode the boardgamemechanic column
exploded_mechanics = cleaned_data['boardgamemechanic'].str.strip("[]").str.replace("'", "").s

# Encode the exploded mechanics into new columns
encoded_mechanics = pd.get_dummies(exploded_mechanics, prefix='Mechanic_', prefix_sep='').gro
encoded_mechanics = encoded_mechanics.replace({True: 1, False: 0})

# Concatenate the encoded mechanics with the original dataframe
cleaned_data_encoded = pd.concat([cleaned_data_num, encoded_mechanics], axis=1)


# Split the data
X_train_full, X_test_full, y_train, y_test = train_test_split(X_full, y, test_size=0.2, rando
X_train_reduced, X_test_reduced, _, _ = train_test_split(X_reduced, y, test_size=0.2, random_

# Standardize the data
scaler_full = StandardScaler()
X_train_full_scaled = scaler_full.fit_transform(X_train_full)
X_test_full_scaled = scaler_full.transform(X_test_full)

scaler_reduced = StandardScaler()
X_train_reduced_scaled = scaler_reduced.fit_transform(X_train_reduced)
X_test_reduced_scaled = scaler_reduced.transform(X_test_reduced)
```

```
# Train Ridge regression for the full model
alpha_full = 0.1  # You can adjust the alpha parameter
model_full = Ridge(alpha=alpha_full)
model_full.fit(X_train_full_scaled, y_train)

# Train Ridge regression for the reduced model
alpha_reduced = 0.1  # You can adjust the alpha parameter
model_reduced = Ridge(alpha=alpha_reduced)
model_reduced.fit(X_train_reduced_scaled, y_train)



# Generate predictions
y_pred_full = model_full.predict(X_test_full_scaled)
y_pred_reduced = model_reduced.predict(X_test_reduced_scaled)

# Calculate R-squared
r2_full = r2_score(y_test, y_pred_full)
r2_reduced = r2_score(y_test, y_pred_reduced)

# Calculate F-statistic and p-value for the full model
n_full = len(y_test)
p_full = X_test_full_scaled.shape[1]  # Number of predictors
f_stat_full = ((r2_full / (1 - r2_full)) * (n_full - p_full - 1)) / p_full
p_value_full = 1 - f_statistic.cdf(f_stat_full, p_full, n_full - p_full - 1)

# Calculate F-statistic and p-value for the reduced model
n_reduced = len(y_test)
p_reduced = X_test_reduced_scaled.shape[1]  # Number of predictors
f_stat_reduced = ((r2_reduced / (1 - r2_reduced)) * (n_reduced - p_reduced - 1)) / p_reduced
p_value_reduced = 1 - f_statistic.cdf(f_stat_reduced, p_reduced, n_reduced - p_reduced - 1)

# Print the report
print("Regression Report:")
print("==================")
print("Full Model:")
print(f"R-squared: {r2_full}")
print(f"F-statistic: {f_stat_full}")
print(f"p-value: {p_value_full}")
print("\nCoefficients:")
for feature, coef in zip(X_full.columns, model_full.coef_):
    print(f"{feature}: {coef}")
```

```
print("\nReduced Model:")
print(f"R-squared: {r2_reduced}")
print(f"F-statistic: {f_stat_reduced}")
print(f"p-value: {p_value_reduced}")
print("\nCoefficients:")
for feature, coef in zip(X_reduced.columns, model_reduced.coef_):
    print(f"{feature}: {coef}")



# Get coefficients for the full model
coefficients_full = pd.DataFrame({'Feature': X_train_full.columns, 'Coefficient': model_full
coefficients_full['AbsoluteCoefficient'] = abs(coefficients_full['Coefficient'])
coefficients_full_sorted = coefficients_full.sort_values(by='AbsoluteCoefficient', ascending=

# Get coefficients for the reduced model
coefficients_reduced = pd.DataFrame({'Feature': X_train_reduced.columns, 'Coefficient': model
coefficients_reduced['AbsoluteCoefficient'] = abs(coefficients_reduced['Coefficient'])
coefficients_reduced_sorted = coefficients_reduced.sort_values(by='AbsoluteCoefficient', asc
# Separate coefficients for Mechanic_ features and other features for the full model
mechanic_features_full = coefficients_full_sorted[coefficients_full_sorted['Feature'].str.sta
other_features_full = coefficients_full_sorted[~coefficients_full_sorted['Feature'].str.start

# Separate coefficients for Mechanic_ features and other features for the reduced model
mechanic_features_reduced = coefficients_reduced_sorted[coefficients_reduced_sorted['Feature
other_features_reduced = coefficients_reduced_sorted[~coefficients_reduced_sorted['Feature']

# Print table for Mechanic_ features for the full model
print("\nFull Model Coefficients for Mechanic_ Features (sorted by absolute value):")
print(mechanic_features_full)

# Print table for other features for the full model
print("\nFull Model Coefficients for Other Features (sorted by absolute value):")
print(other_features_full)

# Print table for Mechanic_ features for the reduced model
print("\nReduced Model Coefficients for Mechanic_ Features (sorted by absolute value):")
print(mechanic_features_reduced)

# Print table for other features for the reduced model
print("\nReduced Model Coefficients for Other Features (sorted by absolute value):")
print(other_features_reduced)
```