Worthwhile Purchase Of PC-Games

A model that predicts if a selected game from Amazon is worth buying, according to customer reviews

Idan Montekyo

Amazon.com Inc. - one of the Big-Five companies in the U.S, is an American technology company which focuses on e-commerce, cloud computing, and artificial intelligence.

Amazon was founded at 1994, originally as an online marketplace for books. But fast enough, started selling discs, movies, electronics, clothes and more.

In this project, we will focus on PC-compatible Games.

Source: Wikipedia

Which one of us have not bought online? Or at least thought of it…

Buying online can be a difficult task.
You can never fully trust an online seller, and nobody promises you will get exactly what you see. Also, is the price really worth it ?

What if there was a model to predict your satisfaction before you actually pay ?
Based on the product’s features - predicting whether the product is worth the purchase or not.

Planning

What phases will the project include ?

1. Data acquisition (5)
   - Crawling and scraping - using Selenium, BeautifulSoup, urllib
2. Data Handling (11)
3. EDA (Visualizations) (12)
   - Pie Charts, Boxplots, Crosstabulations, Scatterplots, Pairplots
4. Statistic Tests (18)
   - Chi-Square test
5. Machine Learning (20)
   - Logistic Regression, SVM, KNN, Adaboost, Naïve Bayes, Decision Tree, Random Forest
   - f1-score, accuracy
   - Confusion Matrix
6. Results and conclusions (21)

Data Acquisition

The data have been withdrawn from Amazon.com.
I focused on PC-Compatible Games.

Libraries used for the acquisition phase:

Selenium
requests + User-Agents
urllib + User-Agents
bs4 (BeautifulSoup)

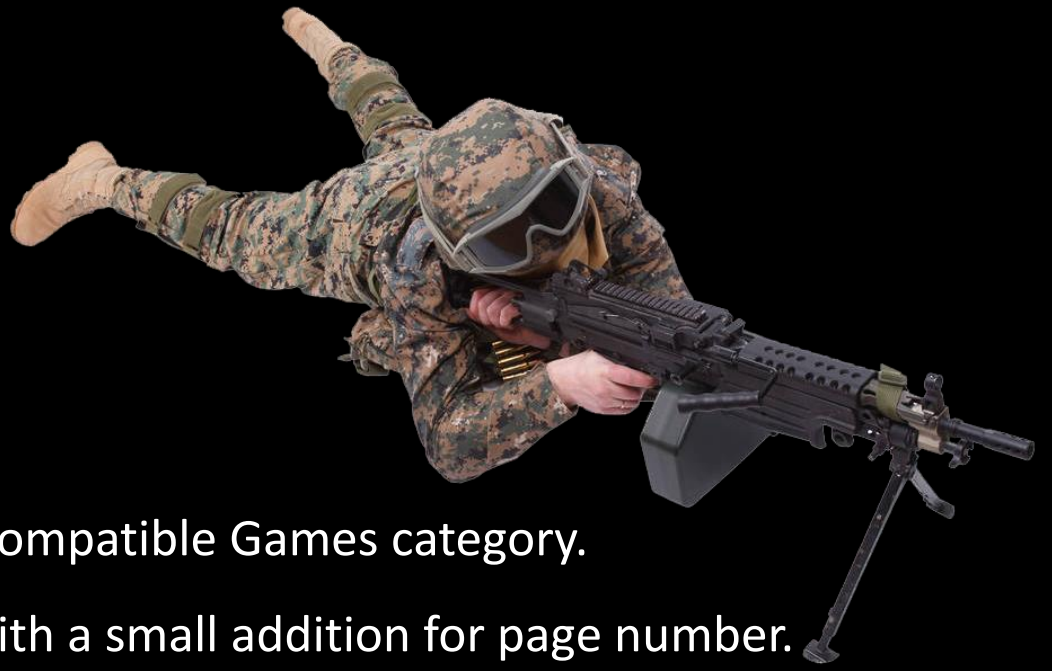
re
time
pandas
os

Data scope:

2428 products (games only)
24 features
Total of 58,272 data

! Note that Amazon have changed their entire website just a few days after I've finished.
As it is right now, my code will not work on their html structure.

Crawling



Step 1

First, we need access to all the games under PC-Compatible Games category.

I noticed that all pages had the same url prefix, with a small addition for page number.
Thus, I decided to get all 400 pages ‘manually’ with a simple for-loop.
Then I added the page links to a list, and saved as a csv file.

Crawling

Step 2

Extract all the links from all 400 pages under PC-Compatible Games category.

⚠ This process was done with ‘requests’, ‘urllib’ and ‘bs4’ libraries.

Note that before starting, we had to examine our site.
I’ve noticed that each page holds 20-22 games, but only 16 are actual games for the page.
Each page had advertisements of sponsored games that repeat for the same set of games over and over.

Filtering these games by searching for the word ‘sponsored’ in each game-section in the source-code,
⚠ we managed to filter thousands of duplicates at such an early stage.

Crawling

Step 3

Now we had to explore each game page and extract it's features.

Note again, that this time we had to do another filtering stage.
It turns out that the 'PC-Compatible Games' category holds A LOT of non-game products, such as headphones, controllers, mouses, VR-headset, and much more.
I searched for a common feature, and noticed that only games had a 'Genre' category.

So this time the crawling was done for the game list, and the condition was that the product must have the word 'genre' in it's source-code.
Only then, the page scraping process began, and data was collected.

⚠ With this step, we managed to filter tens-of-thousands of duplicates at such an early stage.

⚠ Due to Amazon blocking me time and time again - this process was done with 'Selenium' library.
This way, using Selenium to get the source-code, I managed to avoid getting <Response [503]>.

Crawling

Technical notes and problems

During the process, I've faced multiple problems.

⚠ <Response [503]>
For a really long time, I was stuck on the request-response part.
Apparently, Amazon's website is really sensitive, and after a few requests I got blocked again and again.
It took me a while to solve this problem.
First, I've tried sending requests with requests library. Then I've tried adding headers as User-Agents.
Then, I've tried using urllib library, and even tried combining the two with Try-Except conditions.
But finally I understood that the only way to bypass this problem is by using Selenium.

⚠ Also, because of the fear of getting blocked, the data was automatically re-saved after every 200 games.

⚠ The site holds too many non-game products. Thus, data was enough, but not as much as expected.

⚠ The scraping process required a few uses of Regular Expression searches, to get the requested data.

⚠ Early assignment of 0 / -1 values to missing data, to avoid NaN values in our data set.

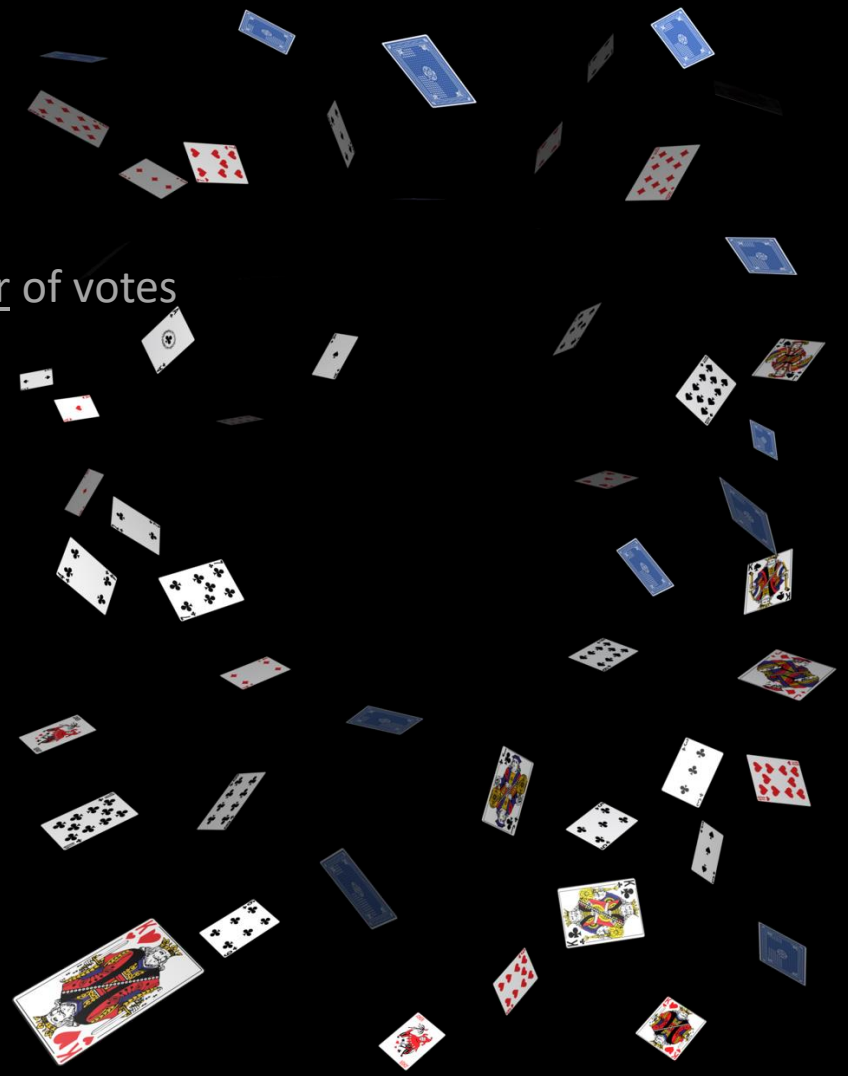
My Data Frame

My Dataset is composed of 24 features:

- str Name
- float Price + Discount - in USD, products with no price assigned -1
- float Rating - 1 to 5, products with no ratings feature assigned -1
- int Number-of-ratings
- int X-star - what percentage of the votes was for X stars, converted to number of votes
- str Brand
- str Genre + Sub-genre - if one - takes it, if more - takes the first two
- str -> int Publication-date - split to day, month, year using Regex
- str Operating-system + Sub-operating-system - same as Genre
- str Format
- str Platform
- str Other-features - checks if the product has another feature
- str Description
- int Description-bullets-num + Description-size

  * Missing data (except for prices/ratings were assigned as 0)

⚠ Usage of Regex was required for prices, ratings, and dates scraping.



Data Handling

Remove all duplicated products

Convert months to numbers 0-12 (0 for missing data)

Convert Other-features to a binary column, whether there is an extra feature or not

Convert all Object-type features to categories represented as Integers

⚠ * Included using various forms of Regular Expressions and manipulations.

Create a binary ‘Worthy’ column, to classify whether or not the product is worth buying

Option to assign mean value to missing prices - or - drop these products

* Spoiler - we are going to remove them - explanation later in the presentation.

Examine and remove outliers

Remove all products with missing rating

* A nice addition might be coming back to this part in the future, to see if we can predict an answer for these games too.

Set ‘Name’ column as the data-frame’s index

Visualizations

Pie charts

Will help us determine how to set ‘Worthy’ based on Rating distribution

Boxplots

Will help us detect values suspected as outliers

Scatterplots

CrossTabulations

Pairplots

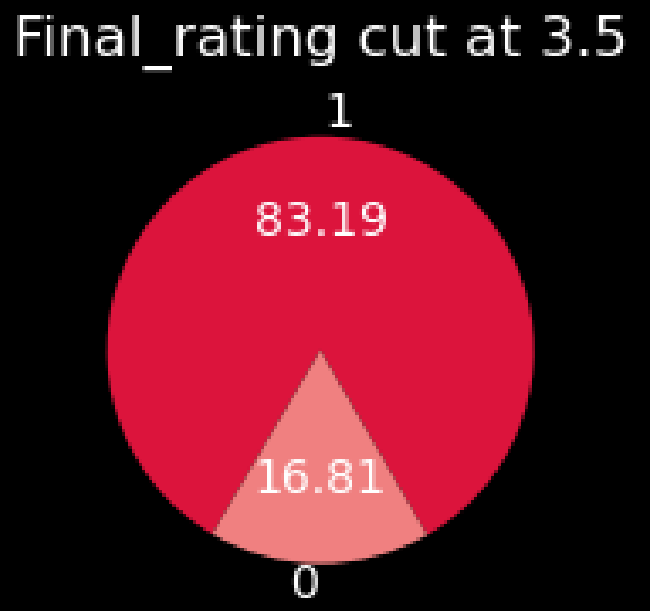
Will help us understand the relations and dependencies between features

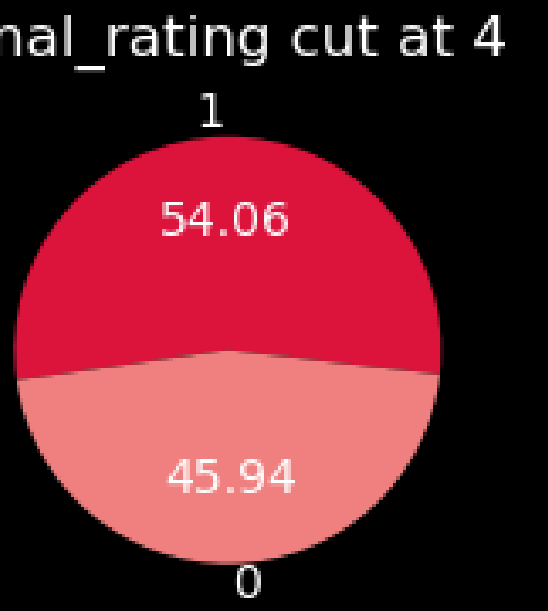
Pie Charts

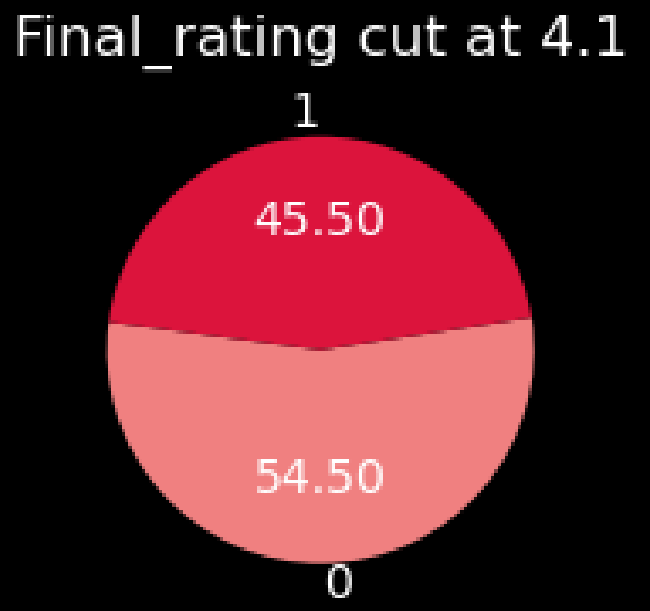
These pie charts show us the distribution of ‘Final_rating’ column.

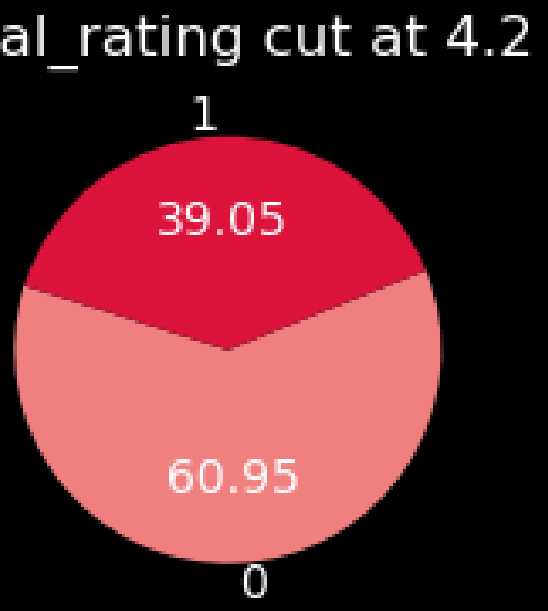
It is important to split wisely and get a balanced division.

Following these pie charts, I decided to split at the rating of 4 in order to make the binary ‘Worthy’ column.









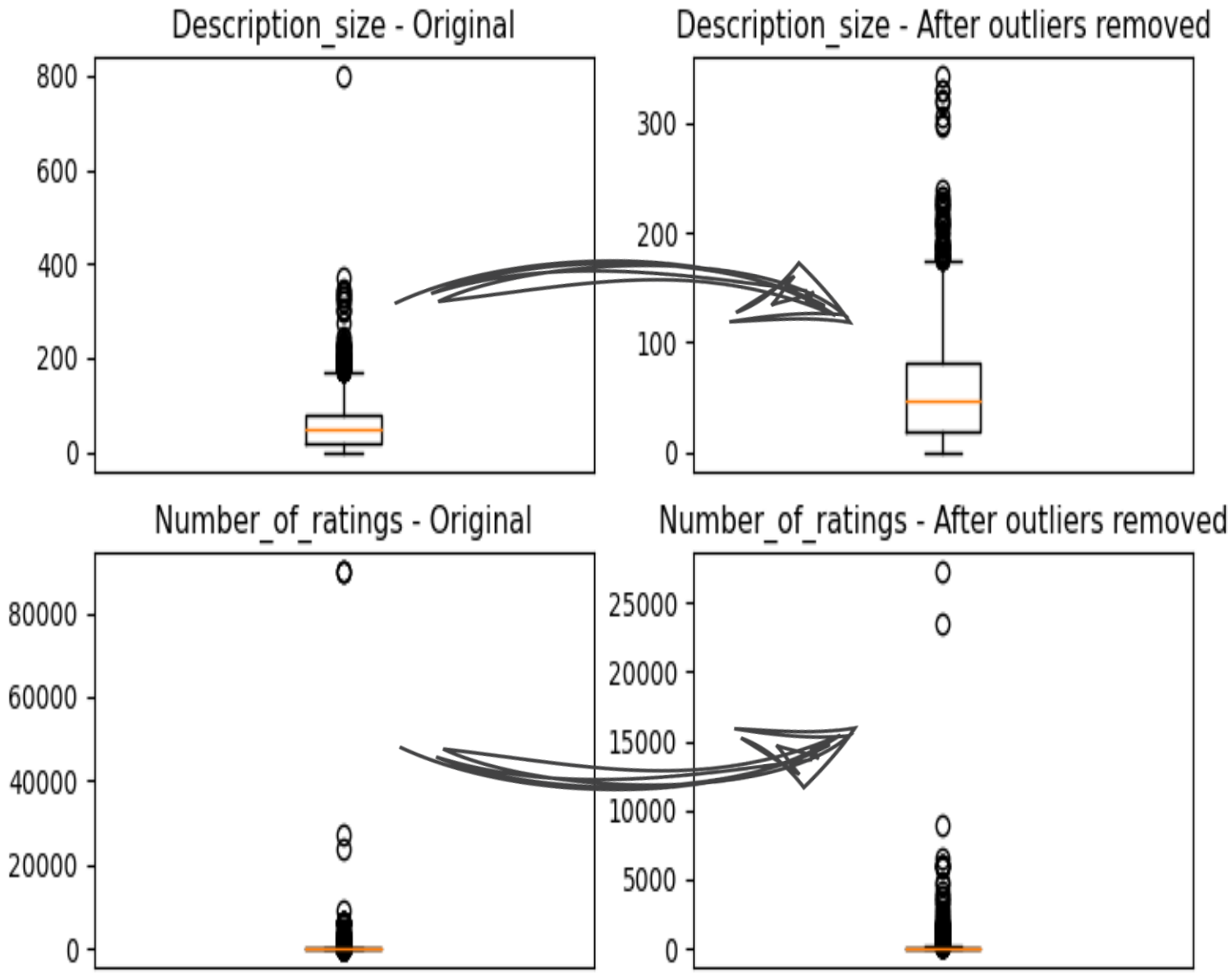
Boxplots

The boxplots can help us detect outlier values.

I used it to find suspected values, then inspected them manually to understand if they make sense or need to be removed.

Note that the decision to remove suspected values is up to the data-scientist.

For example, the boxplots classified two games with extremely high prices as outliers, but after inspecting the games - the prices were verified, thus I did not remove those products.



Scatterplots

Examine relation and dependency of ‘Worthy’ with various features.

We can infer that products with relatively large amount of 5 stars or 1 star rates will have better chance of being worthy to buy.

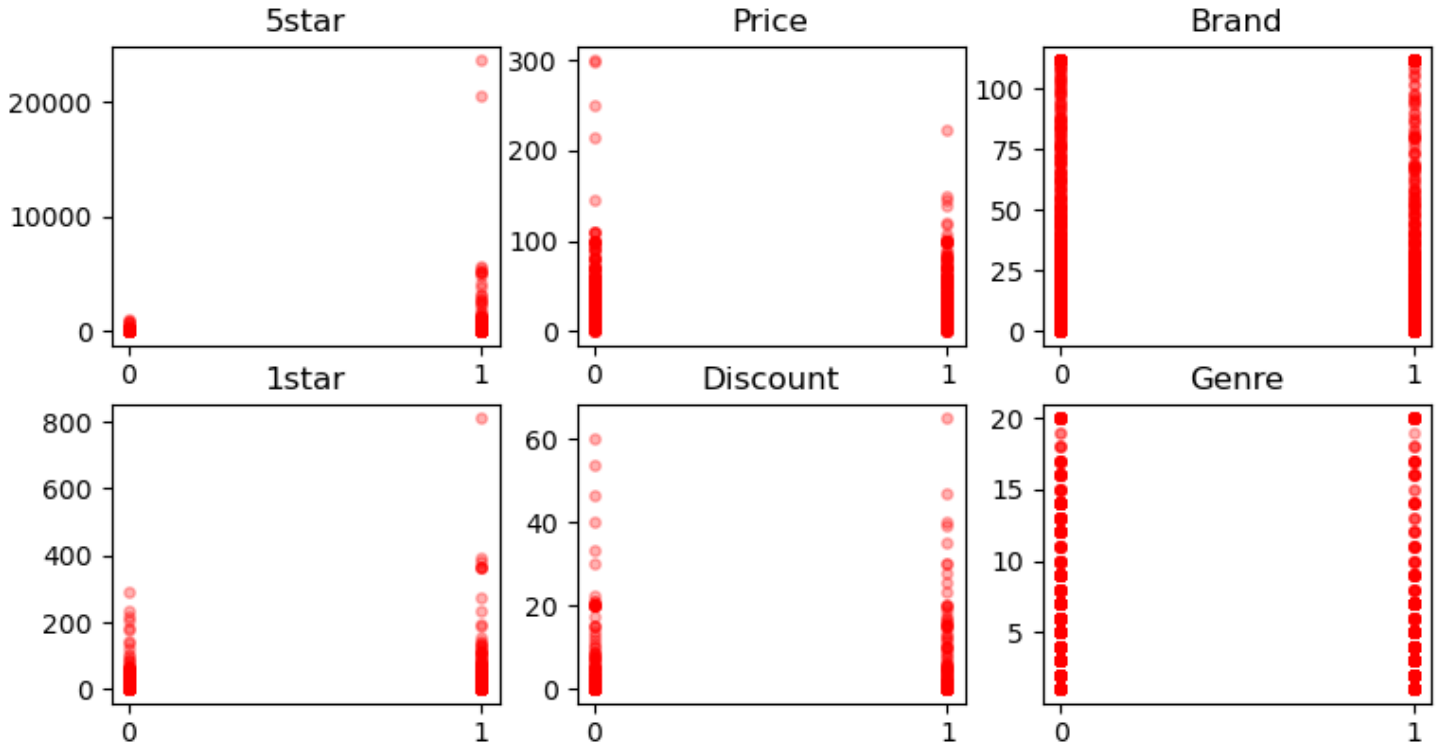
While products with relatively high price have less chance of being recommended.

* X-stars will obviously not be part of the learning phase due to them being part of what we want to predict.
But they do present a fine idea of how the Scatterplot works.

Large range of values

Floats

Categories



Crosstabulation

Examine relation and dependency of ‘Worthy’ with various features.

We can infer that products with an extra feature will have better chance of being worthy to buy.

We can also see that games from 2020 on are more likely to be recommended comparing to the previous 10 years.

Discrete values or a binary feature

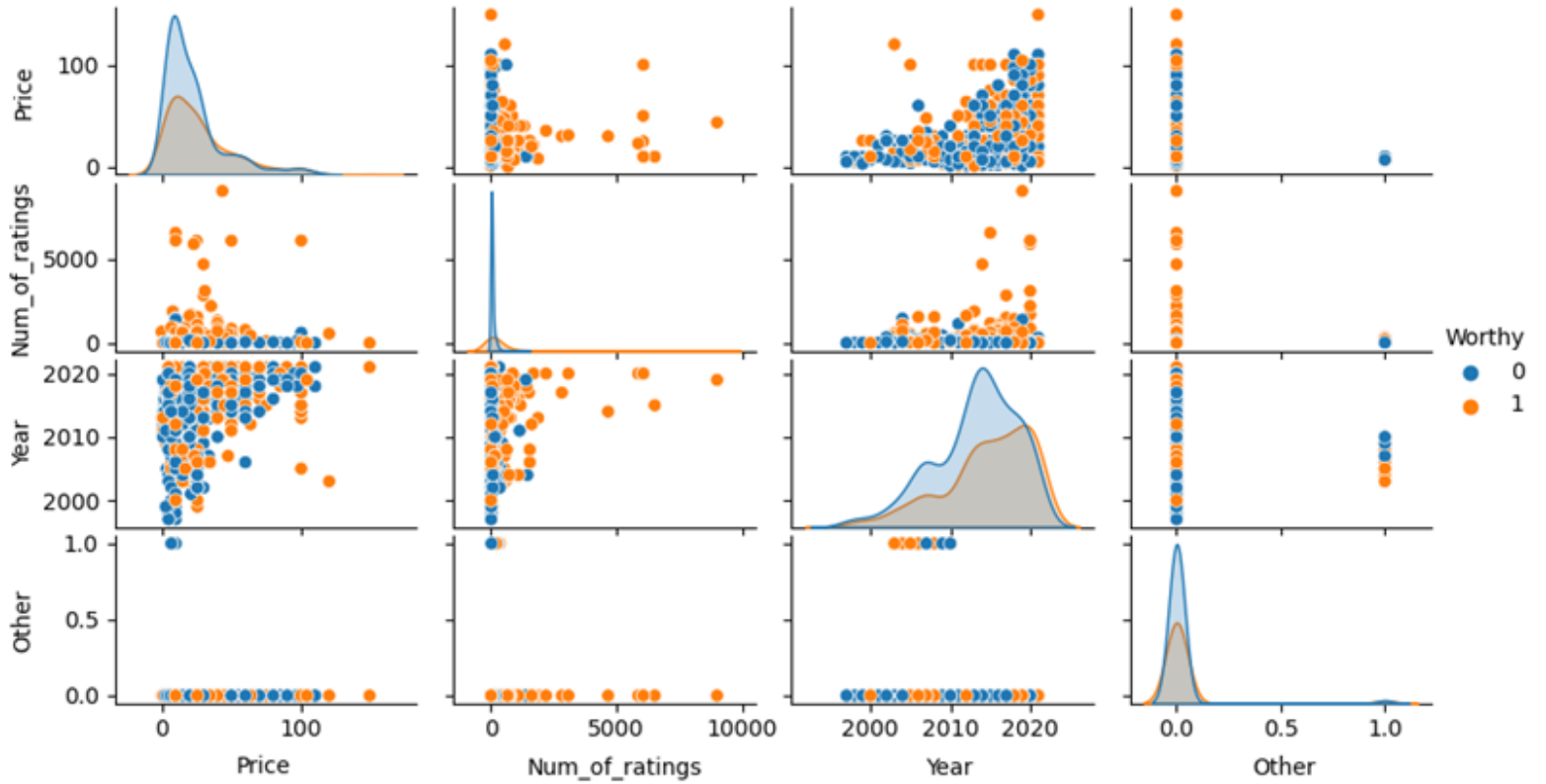
Pairplot

Examine relation and dependency between pairs of features, while marking the consequences on ‘Worthy’ with various features.

We can see at [2, 1] that newer games have much higher number of ratings, and also are likely to be worth the purchase.

At [2, 0] we see that newer games are likely to cost a lot more than old games.

And at [1, 0] we see that cheaper games have much higher number of ratings, meaning there is probably more demand.



Statistic Tests

Chi-Square test is a statistical test that examines dependencies.
I used it to test the dependencies between my target feature ‘Worthy’ and most of my dataset’s features.

If the result < 0.05, the features are considered dependent.
We can see that ‘Final_rating’ is obviously 0 (True) because ‘Worthy’ is actually based off of that feature.

To my surprise, it seems that ‘Worthy’ depends on ‘Price’ and ‘Discount’ features much less than anticipated.

```
Chi-Square test for dependency: Test approved if Result < 0.05
Dependency in feature - Price: False -> 0.060474452282018l7
Dependency in feature - Discount: False -> 0.21676020430381354
Dependency in feature - Final-rating: True -> 0.0 (Obviously...)
Dependency in feature - Number-of-ratings: True -> 1.0845835633459083e-13
Dependency in feature - 5-star: True -> 2.7793519912054865e-15
Dependency in feature - 1-star: True -> 1.7922295715007 79e-22
Dependency in feature - Brand: True -> 1.7986873255460 08e-16
Dependency in feature - Genre: True -> 2.0784529677124487e-08
Dependency in feature - Month: True -> 0.022920783540046202
Dependency in feature - Year: True -> 3.7680982682065274e-06
Dependency in feature - Operating-system: True -> 7.281919241606231e-12
Dependency in feature - Format: True -> 0.002334260159655363
Dependency in feature - Platform: True -> 0.0010855671880885O7
Dependency in feature - Other-features: True -> 0.00341971813756468 44
Dependency in feature - Description-bullets: True -> 0.022889777711041 35
```

Run TODO Problems Debug Python Packages Terminal Python Console

Feature Engineering

Manipulations on the dataset included:

X-star was converted from percentage of votes out of the total votes - to number of votes.

Converting publication date to day, month and year.
Extracted from a single string using Regular Expressions.
Then converted month from string to integers.

Values suspected as outliers were examined. Some were removed.

Name feature was assigned as dataframe's index.

Features representing categories were converted to integers using Regular Expressions and other manipulations.

Other-features changed into a binary feature, whether there is another feature or not.

Discount, Month, Rating, X-star, Description were removed for Machine Learning phase.
Number-of-rating remain because it resembles the number-of-purchases, since only buyers are able to rate a product.

All games with no Rating feature were removed since we can not learn from them nor confirm prediction on them.

Created new binary 'Worthy' column, representing whether a product is worthy to purchase based on users ratings.

After confirming it will upgrade the learning results - all games with no Price feature were removed also.

Machine Learning

My dataset's label is 'Worthy' because my research question is whether or not a game is worthy to purchase.

As already mentioned, after testing the full dataset vs. the reduced dataset, I continued with the reduced dataset due to better performance.

After examining several results, I came to the conclusion that it's best to divide the data into 70% training, and 30% testing in this case.

I've tried improving the model by scaling the data (StandardScaler, MinMaxScaler).
The results were worse, so I decided to move on without scaling.

I've tried using several learning algorithms: Logistic Regression, SVM, KNN, Naïve Bayes, Decision Tree and Adaboost.
But the best results were for Random Forest algorithm.

Finally, after feature engineering stage I managed to get the result of over 75% success.

Results & Conclusions

Filling missing prices
vs.
Removing missing prices

```
Testing which dataset is
more accurate and better to use:

No-price products assigned
mean of existing-price products:
0.611904761904762
0.6095238095238096
0.6023809523809524
0.5928571428571429

No-price products removed,
left with existing-price products:
0.6352941176470588
0.6431372549019608
0.615686274509804
0.615686274509804
```

Testing several learning algorithms
Output: accuracy, f1-score, confusion-matrix

```
Logistic Regression
Accuracy: 254 / 382  ->  0.6649214659685864
f1-score: 0.6893203883495145
[[112  53]
 [ 75 142]]

SVM
Accuracy: 229 / 382  ->  0.599476439790576
f1-score: 0.48484848484848486
[[157   8]
 [145  72]]

KNN, n = 11
Accuracy: 248 / 382  ->  0.6492146596858639
f1-score: 0.6839622641509434
[[103  62]
 [ 72 145]]

Naive Bayes
Accuracy: 203 / 382  ->  0.5314136125654451
f1-score: 0.3443223443223443
[[156   9]
 [170  47]]

Decision Tree
Accuracy: 221 / 382  ->  0.5785340314136126
f1-score: 0.6229508196721313
[[ 88  77]
 [ 84 133]]

Adaboost
Accuracy: 270 / 382  ->  0.7068062827225131
f1-score: 0.7358490566037736
[[114  51]
 [ 61 156]]
```

Results & Conclusions

In this case, Random Forest was the best algorithm.

Regardless the max-depth parameter, the results surpass all others.

After delving deeply into the two best results, I decided the best solution for this research would be Random Forest with max depth of 5.

Although it does not have the highest f1-score,
A simple look at the confusion-matrix tells us it is the most balanced result.

⚠ Note that the Feature Engineering phase contributed A LOT and raised our model's success rate and accuracy.

```
Random Forest, max-depth = 2
Accuracy: 276 / 382  ->  0.7225130890052356
f1-score: 0.7633928571428571
[[105  60]
 [ 46 171]]
```

```
Random Forest, max-depth = 5
Accuracy: 277 / 382  ->  0.725130890052356
f1-score: 0.7368421052631579
[[130  35]
 [ 70 147]]
```

Results & Conclusions

True Positive
The model classified this game as worthy

True Negative
The model classified this game as not worthy