

Unlocking the Forge: Using Machine Learning to understand the game Keyforge.

Higher Diploma in Science in Data Analytics

Word Count: 1551

Acknowledgments

Barry Sheppard

Barry.Sheppard@gmail.com

23/10/2019

Supervised by Paul Laird

Paul.Laird@dbs.ie

Contents

[Contents 3](#_Toc26825156)

[1. Introduction 4](#_Toc26825157)

[2. Background 5](#_Toc26825158)

[3. Requirements Specification and Design 5](#_Toc26825159)

[4. Stage 1 – Data extraction 6](#_Toc26825160)

[5. Stage 2 – Data cleaning 7](#_Toc26825161)

[6. Stage 2 – Data Analysis 8](#_Toc26825162)

[1. Stage 3 – Recommender 10](#_Toc26825163)

[2. Future Work 10](#_Toc26825164)

[3. References / Bibliography 11](#_Toc26825165)

# Introduction

KeyForge (“KeyForge,” 2019) is a collectable card game developed by Fantasy Flight Games (“Fantasy Flight Games,” 2019), similar to Pokemon (“Pokémon Trading Card Game,” 2019), Yu-Gi-Oh (“Yu-Gi-Oh!,” 2019), and Magic the Gathering (“Magic: The Gathering,” 2019). Where it sets itself apart is in the sales model. Rather than selling players of the game random cards, which are then combined to create decks, the company sells pre-generated random decks which cannot be changed. These decks are all unique meaning no two decks contain the same combination of cards. This retail model means a player can buy a single deck and be ready to play. It also means the more dedicated players will buy multiple decks to get the best deck they can (D’Anastasio, 2018).

From a data analysis perspective, this provides a discrete problem to solve. If we were to buy two decks, could we determine which was the better deck? First, we would need some data on which to base our decisions. Luckily, after a player buys a deck, they register it on the companies site (“KeyForge Master Vault,” 2019) which shows a full list of all the cards in that deck. This listing of cards will be our predictor variables. Good cards or card combinations in the deck will improve the chance of the deck winning. Secondly, when players enter into tournaments, the wins and losses of each deck are recorded. So we have an outcome variable, decks with higher win rates have demonstrated better performance than those with low ones.

The Aim of the project is to extract this data from the company’s website, clean that data for use, determine the best machine learning model, we can then create a predictor to select the best deck from those presented. The scope of this project is narrow due to the available data. While there are over 1.25 million registered decks, the info available on those decks is minimal. We merely know the 36 cards in each deck and the number of wins and losses during tournaments. We do not know the skill of the players involved, the power of the deck's played against, or how lucky the player was in their draw of the random cards. Based on that, the project scope and approach are similar to most data analysis projects.

1. Extract the data from the website using Python saving it to file.
2. Clean and structure the data for analysis using Python functions.
3. Create descriptive plots and detail any insights based on the data in Jupyter Notebook.
4. Generate machine learning models based on the data.
5. Test and improve the models.
6. Using the final model, create a program that will select the best deck from a range.
7. Detail any insights developed from the model.

This is based on the assumptions that the data can be easily extracted from the source website and that meaningful insights can be drawn from the data.

# Background

As the focus of this project is on a casual game, the context is relatively light. Understanding which type of deck performs better will allow players to make better decisions when selecting decks. Although this particular game does have some cash prize tournaments and a secondary sale market around decks, largely it is casual game played just for fun.

Given the data provided, the analysis is completed on the assumption that decks which perform better do so because of the component elements of the deck. In a real world context, however, games can be won by lucky draws or player skill. Similarly, the data shows the wins and losses for each deck but doesn’t give details of the opponent. This means, a deck that has a high number of wins could be played only versus poorer opponents while another deck that has a low number of wins could have been played against only the best decks available. For this project, these uncontrolled variables are being considered random noise, but it is important to acknowledge their existence.

Based on the current scope, the expected technical requirements is minimal with the initial extract completed with a Python script and the data analysis element completed in Jupyter Notebook (“Project Jupyter,” 2019) or Google Colab (“Google Colab,” 2019).

# Requirements Specification and Design

The core requirement for this project is to provide a way to assess the competitiveness of a deck. Preferably, this should allow us to determine which deck is better when given any two decks, but if this level of grain is not achievable, at least grading a deck is desired.

To generate an algorithm than can do this, we require lists of existing decks and their performance in tournaments which includes their wins and losses. As any deck can be registered on the website, the page for the deck can be provided as an input to be assessed by the algorithm. In addition, a written report may provide some high level decision points, such as key cards to look for, which could be used by player without requiring machine assistance.

At a top level, the desired data analysis workflow will begin with a snap shot of the existing data extracted from the system. That data will then be cleaned and prepared for analysis. Once ready, the data will be used to developed algorithms that can be then used to predict the performance of a given deck.

From a develop perspective, one goal of this project is for the author to develop skills with Python in the full end to end workflow. While a similar process was developed during the course, the focus for these elements primarily used the R programming language.

# Stage 1 – Data extraction

The proposed project timeline anticipated having data extracted and cleaned by the 23rd of October with some initial insights generated. The initial assumption was that the data could be quickly extracted through the API allowing more time to focus on analysis, but as often happens in projects like this, that was not the case.

The source website did provide an API, although it is undocumented. Some useful information was sourced from another developer’s attempt to use the API which provided a starting point as it detailed some of the options and outputs if not actual code. This provided a number of API hooks but presumably not the full suite. The API returns a JSON object which includes information about the decks and if requested, additional information about the cards. Extracting the required information from a JSON object was straightforward but was complicated slightly by the way the API returned two separate JSON objects in the same request, one for deck and one for cards. Once a single instance had been successfully processed, the next step was to create the load.py file which would collate multiple instances and combine into a single dataframe. For this, a Pandas dataframe was generated with each new item appended to it.

On attempting this, after just a few runs there was an error as the website’s flood protection kicked in. To prevent flooding the site with requests, after some experimentation, a 60 second delay was added after every 5 items. An additional check was also implemented with a ‘try except’ check which on error delayed the process for 60 minutes before trying again. This successfully avoided the error but resulted in a significant delay reducing the rate to 5 decks every minute. After reviewing the API again it was determined that up to 25 decks at a time could be downloaded at a time in a single request, but when doing so only a single list of cards was generated which covered all instances across the 25 decks. To accommodate this, functions were re-written and the data was split into two dataframes one for the decks which included a list of the card\_ids for each card present and another dataframe which included all the information on each card. During analysis, these dataframes could be referenced as needed to get a full picture.

To facilitate bulk extraction, the load.py file was structured such that a start number and end number were specified. When run the code would create a new blank .csv file for decks and one for cards with a name based off the numbers (e.g. cards\_10001\_12827.csv). Due to the 60 minute delay, these extracts were run over an extended period typically taking several hours to fully complete. As other API issues could cause the code to stop, the code appends each row to the .csv making and reports that last completed number such that any issues will still return a valid dataset up to that point. File names could then be renamed as required.

These issues are reflective of the API design which in retrospect does not appear to have been fully developed for external use yet. The poor performance of the API has been noted by some other developers and is discussed in further detail below. Data collection was paused at the end of October 2019.

# Stage 2 – Data cleaning

At the start of this stage, the data has been saved in a folder for decks and a folder for cards each containing multiple .csv files each with several hundred entries. This data cleansing process was completed in Jupyter Notebook which allowed inspection of the data during the coding process to reveal any potential issues. The first step was to consolidate the files into a single .csv. As files had been saved into their relevant folders a os.listdir was used to select all files in the folder and if i.endswith("csv") was used to only select the csv files. These were then combined into a pandas dataframe for all the cards entries and another dataframe for all the deck entries. From the file a total of 1248450 decks and 15081358 cards. Out of the decks, the majority were never played in a tournament and as such have no wins or losses recording against them. Without these records the decks are essentially unrated and the best deck will look the same as the worst deck, as such we remove all without at least 1 win or loss leaving a total of 67321 decks. Looking at the list of cards, as each deck listing included all cards in it there were a lot of duplicates involved. After removing duplicates this is reduced to 3636 cards. At this point, both dataframes were saved as .csv files to minimise the repetition during the processing. A frustration revealed later in the process was the addition of blank columns, there were eventually revealed to be an index column added with every export to csv. To avoid this in future index=False was added to the calls of the .to\_csv function.

The next stage was to prepare the data for analysis. The main part of this was to take the columns which originally contained lists and convert them into columns. The first column contained all the ‘house’ names, each card belongs to a particular house and each deck has 3 different houses out of a total of 7 houses. As the data had been saved to a csv file at this stage it was in a string format, so the str.contains function was used to determine a Boolean value for each deck. Initially, a row by row process was designed but this resulted in very slow process, after some reading it became apparent that a column by column process was far more efficient when using pandas dataframes. After running this piece of code an extra column for each house was added with a value of True or False for each deck.

A similar process was then completed for cards. As there are a far larger number of cards than there are houses, this process took longer. The first step was to determine the unique names of each card, this was done by converting the list of names into a dictionary which immediately removed all duplicates and then restoring it back to a list. On review, there were 574 unique card names each of which could appear multiple times in a deck. Due to the structure of the game, each card comes from a house, however, on rare occasions a card will appear as a ‘maverick’ card. A maverick card is one which turns up in another house. So each card in theory could appear in any of the houses which explains why the unique names of 574 are much less than the unique cards of 3636. The card dataframe does contain a different card\_id for each of these maverick cards even though the title of the card remains the same. The card\_id is then used, comparing the card\_id from the deck with the card\_ids from the card database to return the name of each card. As each listing of cards maintains a regular format of 36 characters per card\_id starting, 1440 characters, and 36 cards, the string can be cycled through to get all the card ids and return a list of card names. This particular step takes around 90 minutes with the current dataset as it requires a nested loop giving it O(n2) complexity.

With a list of card names the next step is to convert these to columns. The original plan had been to structure the data in a long format with a single row for each card and then pivot it into a wide format but after lessons learned from earlier steps, a column based approach was instead used. Rather than using str.contains which would return a Boolean, this time str.count was used which resulted in a count of the number of cards of that title in each deck. This process was taken for each column, cycling through the full list of all 574 card names. During this process, an error that did appear related to the card name ‘[REDACTED]’ which was not a useable column title due to the square brackets.

To have target variable we calculate a score variable by taking the number of wins of a deck and subtracting the number of losses. This value should give a rough metric of a decks performance. Other approaches were considered, such as calculating a percentage value where the number of wins were divided by the total number of games reported but this would leave a deck with 100 wins and no losses at the same score as a deck with only 1 win and no losses.

A final amendment was made to change the expansion column, which contained either 435 or 341, into two Boolean columns indicating whether the deck was from the ‘Call of the Archons’ first expansion or the ‘Age of Ascension’ second expansion. Once all the data cleaning has been completed, the dataframe was saved to a csv file called data\_decks\_cleaned.csv for use in the next stage of the process.

# Stage 3 – Data Analysis

With the data in a useable format, the next stage is to review it to get a better understanding of the data. Using the describe function we can get some basic info on the decks. The max number of wins for any deck is 40 but 75% of decks have wins of 3 or less. Similarly, the max number of losses is 39 but 75% again are 3 or less. Scores range from -33 for the worst rated deck to 26 for the best rated deck, but the interquartile range is quite small going from -1 to 2. A histogram of the deck scores shows that that the majority are at 0 and with wins and losses are between 0 and 5. Unfortunately, this uniformity in the dataset may prove a challenge for an effective model but these are the kind of things that happen when looking at real data rather than an educational dataset.

Looking at the expansions we can see that majority of decks come from the first set ‘Call of the Archons’. As the dataset is sorted by an id reference and that id reference is generated when a deck is registered, this makes sense as for the first few months only the first set was available. When comparing the scores, we can see that the first set has a better overall score total. While this might be reflective of less games played, the score calculation is actually interested the difference between wins and losses, so the difference may instead be reflective of the set power. This would match player speculation which held that the decks in the first set are generally better than the decks in the second set.

The first model attempted is a random forest, this was selected due to the good performance of random forests generally rather than a reason specific to this dataset. The model used the default parameters and took 18 minutes to run. Using a 70/30 train to data split the resulting model only had a 0.06 r-squared value indicating an extremely poor performance. Attempting much simpler version using only the houses was attempted and again had poor performance with an even worse r-squared value of 0.02! On review of the feature importance for this particular model it was evident that Shadows house was the most important feature by far at 0.74 compared to all the other houses which were at best 0.09. From players of the game, the Shadows house is considered the most powerful and sought after house, so despite the poor performance of the model in general it has identified at least one aspect that matches expectations.

On consideration, the data isn't really well suited for a decision tree. There are over 500 variables and only a very small percentage of those are going to have non zero values. For individual trees to have good predictive quality they will have to be very deep and due to the large number of variables there will also be a larger number of potential trees. As random forests using a polling system and it is likely that that there will be more poorly performing trees than well performing trees it is perhaps not a surprise that this doesn’t work well.

It may be that a simple linear model actually gets the best performance. Each of card will end up with a coefficient that could be considered a rating of how good it is. If you draw a card with a high coefficient then you're more likely to win, but if you draw a card with a low coefficient then you're more likely to lose. On this basis, a simple linear regression model was generated again using a 70/30 train and test split. The performance of this model was still quite poor at only 0.09 but that is at least an improvement on the random forests.

After the full dataset was used to generate a model, the coefficients were ordered to give an indication of which cards were the top performers according to this model. At either end, at the top and the bottom, there are some big values but the majority of coefficients are between 34466233 and 34466234 giving a really small range with differences essentially within between 0 and 1 within that range. From experience with the card pool, it is notable that the two cards at the top, Horseman of Pestilence and Timetraveller, have something in common, they both are parts of sets of cards that always appear together. The Horseman is one of a set of 4 cards all types of Horsemen and the Timetraveller is always paired with Help from Future Self. These 4 cards are all at the bottom of the rankings. If we match these cards and average the values out, the Horsemen set comes to 34466233.94 and the Timetravellers come to 34466234.24 and suddenly make sense within the list.

Looking at the rankings the cards are in based on the coefficients and comparing it to some player experience (Gill, 2019) we can see that out of the top ten cards listed in the article we have 5 in our top 10 list. This is especially notable when you consider that there are 582 cards in total! At this stage looking at alternative models to search for better performance is an option, however it may be simply be the case that the data does not support a high accuracy model. For example, a deck that is only played at an event once and won that game will have a score of 1 but might be good enough to win 100 games if more were played! This topic will be discussed in more detail later, but initially we will be accepting this initial model which was saved to file using the pickle library.

# Stage 4 – Recommender

The ultimate goal of this project is to provide a way to decide between a number of decks and decide which was the best choice. To do this, the process of loading, cleaning, and predicting needs to be repeated with just one deck. This was initially completed in a Jupyter Notebook to allow stepping through the process. Once complete, this was recoded as functions in the app.py file which also included a terminal prompt to allow quick input of a decklist and a return of a value.

# Conclusion and reflection

The project continues in

# References / Bibliography

Sheppard, B.T., 2019. barrysheppard/B8IT110-Project [WWW Document]. URL <https://github.com/barrysheppard/B8IT110-Project> (accessed 10.23.19).

D’Anastasio, C., 2018. KeyForge Is A Whole New Kind Of Card Game [WWW Document]. URL <https://kotaku.com/keyforge-is-breaking-the-rules-of-collectible-card-game-1830544174> (accessed 9.10.19).

Fantasy Flight Games [WWW Document], 2019. URL <https://www.fantasyflightgames.com/en/index/> (accessed 9.10.19).

Google Colab [WWW Document], 2019. URL <https://colab.research.google.com> (accessed 9.10.19).

KeyForge Master Vault [WWW Document], 2019. URL <https://www.keyforgegame.com/> (accessed 9.10.19).

KeyForge [WWW Document], 2019. URL <https://www.fantasyflightgames.com/en/products/keyforge/> (accessed 9.10.19).

Magic: The Gathering [WWW Document], 2019. . MAGIC: THE GATHERING. URL <https://magic.wizards.com/en> (accessed 9.10.19).

Pokémon Trading Card Game [WWW Document], 2019. URL <https://www.pokemon.com/us/pokemon-tcg/> (accessed 9.10.19).

Project Jupyter [WWW Document], 2019. URL <https://www.jupyter.org> (accessed 9.10.19).

Yu-Gi-Oh! [WWW Document], 2019. URL <https://www.yugioh-card.com/en/> (accessed 9.10.19).