**Matt Cronin**

**Econ 9000 – Machine Learning – PS1 – Board Game Geek Exercise**

**Data-Scraping**

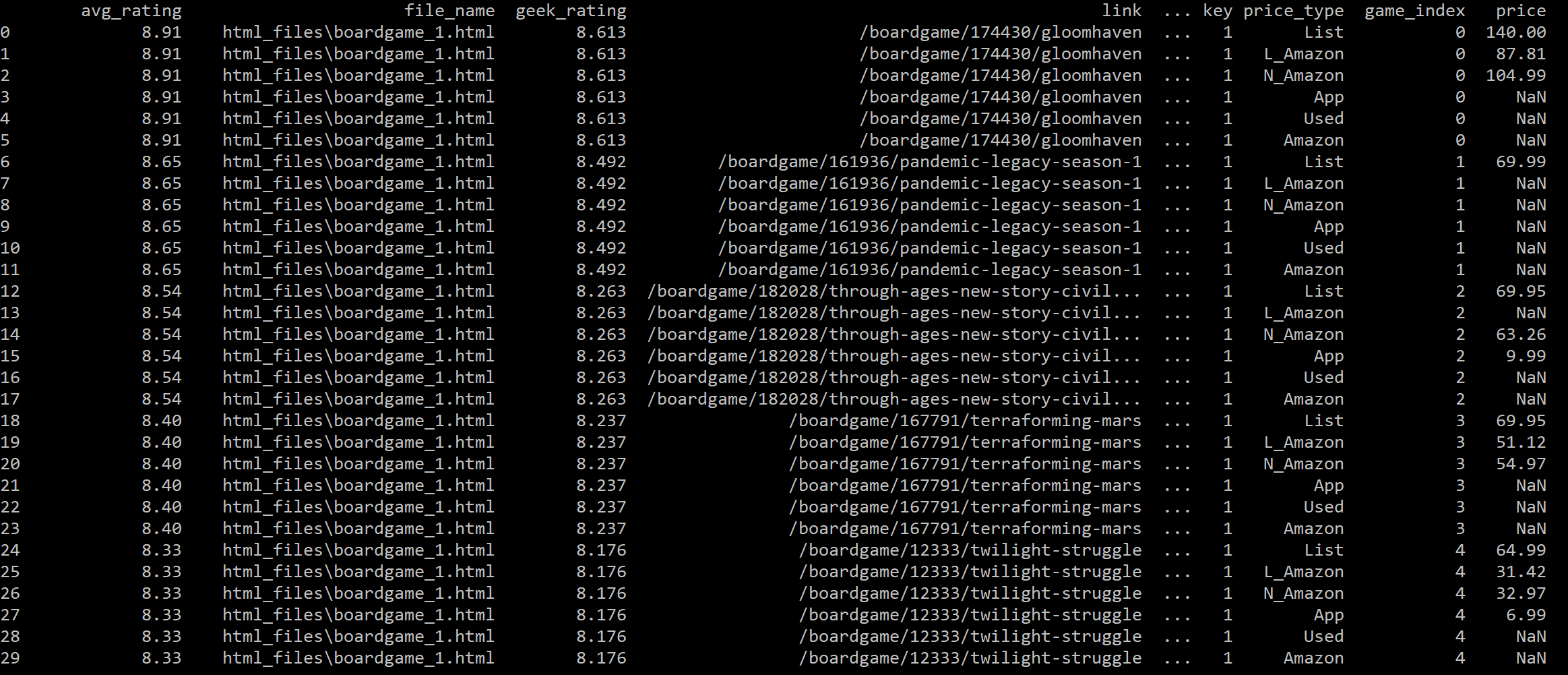
I chose to scrape the data on game names, ratings, number of votes, and prices on boardgamegeek.com. This website contains a table of the game ratings and rankings at the following URL: <https://boardgamegeek.com/browse/boardgame/page/1>

My scrapping program is **‘1\_boardgamegeek\_request.py’**. At the time of scraping this page, these data were contained on tables across 1,059 pages. Since the URL is denotes the page number of each part of the table, a simple loop could iterate through all the pages of the table. I chose to use a while loop to scrape these data that continued to iterate to the next page as long as data was contained on the table. If no data was contained on the next table, then the program would terminate because the end of the table had been reached. Each page was contained on 1059 html files.

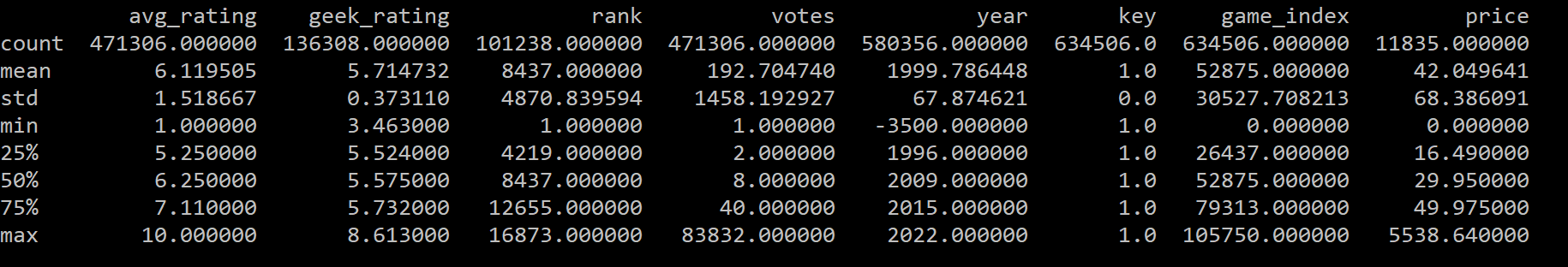
These html files were then parsed using **‘2\_boardgamegeek\_parse.py’**. Data on game name, game rank, game year, geek rating, avg rating, number of votes, prices, and the URL extension for each games information page were extracted. The challenge here was parsing the price data. Since some games have different prices reported, such as “List”, “New Amazon”, “Lowest Amazon”, “Amazon”, “iOs App”, no price, or any combination of the above, I chose to pull all the price text as a string and extract the price data in a later step.

I wrote an additional data building program to clean my game data as well as extract the different prices for each game. **‘3\_boardgamegeek\_pricedata\_clean.py’** cleans the data and splits the price string, described above, into individual observations for each price. The resulting dataframe has rows for each game and price type, where price type is “List”, “New Amazon”, “Lowest Amazon”, “Amazon”, or “iOs App”. If a game did not have a price listed, the price column would be missing for that game and price type.

The first 30 rows of my data frame are here:



The dataset’s numeric values are described below:



**Supervised Learning Exercises**

I performed two supervised learning exercises. They were missing price data imputations and board game geek rating predictions.

**Missing Price Data Imputations:**

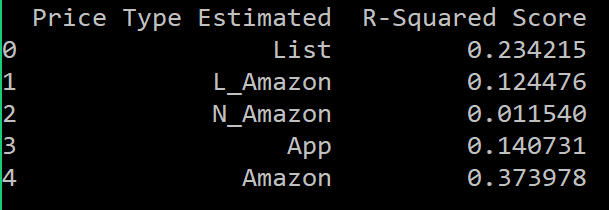
During my data scraping exercise (and as shown in the screenshot of my data frame above), I noticed that not all games had every price type listed. For example, a game could have a list price, but no Amazon prices, or a game could have a new amazon price listed, but no lowest amazon price and no list price. Looking at the first 6 rows of my data frame in the screenshot above, shows that the game “Gloomhaven” has list, new Amazon, lowest amazon, but no app used, or general amazon price (obviously there could be no app version of this game). Going back to board game geek I noticed that the price data they report changes. At some points in time the website will report different price data for the games and new price data becomes available. A researcher looking to perform some analysis on boardgames could go back to the website and re-scrape the data with the hopes that they are lucky and get a more complete draw of prices at that time they decide to re-scrape. However that process is time consuming and the outcome is uncertain in that your new set of available prices may not be a larger set than the prices you have already pulled. In addition to this you re-scrape scrape prices when only a subset of the price data, that you already have, is available. Because only some price observations were missing for certain games, I wanted to perform a supervised learning exercise to impute the missing price data. More specifically I would use the sklearn linear model to estimate 6 regressions and predict each of the 6 price types for each game where that price type price was missing. \_5

To be specific the regression equation I estimated, for a price type *i,* was as follows:

This specification assumes that given a certain average rating, geek rating, rank, and game age a price type *i* can be predicted. After this equation is estimated, prices can be predicted for each price type. The program **‘4\_boardgamegeek\_linearmodel.py’** performs the estimation described above. The general process is as follows:

1. Subset the data frame to a price type *i*
2. Split the resulting data frame from step 1 into the following two data frames:
   1. **Training Data:** Rows with price of type *i* not missing
   2. **Target Data:** Rows with price of type *i* missing
3. Create a linear model machine using the linear regression function in the linear model library from sklearn
4. Use Training Data to perform supervised learning on the linear model machine
5. Use the trained model to predict price *i* for each game in Target Data using the available data on average rating, geek rating, rank, and game age in that data frame
6. Go back to step 1 and perform same process for next price *i* in list of price types

The R-Square scores for each estimation were considerably low indicating that the predictions were not very accurate.



**BGG Rating Prediction:**

The second exercise also used the linear model function from sklearn. This supervised learning exercise was designed to train a machine to predict the geek rating for a particular game. According to boardgamegeek.com, “BoardGameGeek's ranking charts are ordered using the BGG Rating, which is based on the Average Rating, but with some alterations. To prevent games with relatively few votes climbing to the top of the BGG Ranks, artificial "dummy" votes are added to the User Ratings. These votes are currently thought to be 100 votes equal to the mid range of the voting scale: 5.5, but the actual algorithm is kept secret to avoid manipulation. The effect of adding these dummy votes is to pull BGG Ratings toward the mid range.”

In order to predict the BGG rating (given a set rank, average rating, game age, and number of votes), I trained the linear model with the all the available data using the following equation: