Board Game Prediction

Improving Sales Through Data Science

Michael McCann | March 2022

Business Challenge

Attempt to find what makes factors make a good board game. Regression based on ranking, and categorical based on top 100 games (top 100 = True/False). Stretch goal to make a recommendation engine.

Data

Data is derived from two separate Kaggle datasets merged into one larger dataframe for machine learning and a few smaller dataframes for EDA and visualizations.

- Board Game Database from BoardGameGeek: Comprised of nine CSVs looking at various facets of 22k board games (Ratings, Designer, Artist, Publisher, Theme, Mechanics, etc). Accessed via BGG's API.
- Board Game Geek Rankings: Top 5000 games as ranked on Board Game Geek going back to October 2018. Data is scraped weekly from BGG and published to Kaggle.

Combined these datasets based on the games that ranked in the top 5,000 games over the past 2.5 years. This amounted to approximately 6,000 games during that period. Eliminated inconsistencies in the Board Game Database.

Caveats and Considerations

- Board Game Database included a number of obscure games with little data or ratings information.
- Formatting for the ranking data is not ideal and required some work to get into a useable format.

Data Wrangling

Changing the ranking dataframe from one where the rank was the index to where the ID was the index was a major challenge. The loop on the right was the solution I eventually worked out (third/cleaner iteration).

```
1 # Import Data
2 rank_df = pd.read_csv('https://raw.githubusercontent.com/msmcca
3 rank_df.head()
```

	BoardGameRank	2018-10-06	2018-10-13	2018-10-20	2018-10-27
0	1	174430.0	174430.0	174430.0	174430.0
1	2	161936.0	161936.0	161936.0	161936.0
2	3	182028.0	182028.0	182028.0	182028.0
3	4	167791.0	167791.0	167791.0	167791.0
4	5	12333.0	12333.0	12333.0	12333.0

5 rows × 180 columns

```
18 rank_df.head()
```

	BGGId	2019-09-07	2019-09-14	2019-09-21	2019-09-28	2
0	174430	1.0	1.0	1.0	1.0	
1	161936	2.0	2.0	2.0	2.0	
2	167791	3.0	3.0	3.0	3.0	
3	182028	4.0	4.0	4.0	4.0	
4	12333	5.0	5.0	5.0	5.0	

5 rows × 136 columns

```
[13]
# Create pd.Series of all values in columns
game_list = list(rank_df[rank_df.columns[2]])
for col in rank_df.columns[2:]:
    game_list += list(rank_df[col])

game_list = pd.Series(game_list)
game_list.drop_duplicates(inplace = True)
game_list

# Create a new DF using the above Series
new_df = pd.DataFrame(game_list, columns = ['BGGId'] )
new_df.reset_index(inplace = True, drop = True)

# Create a blank dictionary to iterate through the above series
game_dict = {}
for i in list(game_list)[0:len(game_list)]:
    game_dict.update({i : np.nan})

for i in rank_df.columns[1:]:
    temp_dict = dict(zip(rank_df[i], rank_df['BoardGameRank']))
    game_dict.update(temp_dict)
    new_df[i] = new_df['BGGId'].copy()
    # new_df.reset_index(drop = True, inplace = True)
    new_df[i] = new_df[i].replace(game_dict)
    game_dict = {}
    for i in list(game_list)[0:len(game_list)]:
        game_dict.update({i : np.nan})
```

Functions 1

Created function to reverse the OHE data so that I could more easily conduct EDA.
Created a function that took that output and cleaned and melted it for EDA.

[21]
```python
## create OHE reversing function...
def ohe_reverse(row, names):
    # Change the row into a np.array
    cat = np.array(row)
    # Add spaces and commas into the name array for spacing..
    names = [' ' + i + ',' for i in names]
    return ''.join(np.repeat(names, cat))
```

```python
## Create A customer longer function...
def make_longer(data, col):
    # fix output - blanks
    bad_rows = data[data['ohe'] == ''].index
    data.drop(bad_rows, inplace = True)

    data['ohe'] = data['ohe'].str.lstrip()
    data['ohe'] = data['ohe'].str[:-1]

    # Stack the result
    val = pd.concat([data['BGGId'],
                     data['ohe'].str.split(',', expand = True)],
                     axis = 1)

    melt = pd.melt(val, id_vars=['BGGId'])
    long_df = melt[melt.value.notnull()].sort_values('BGGId').reset_index(drop=True).rename(columns={'value':col}).drop(columns = 'variable')
    long_df[col] = long_df[col].str.lstrip()
    return long_df
```

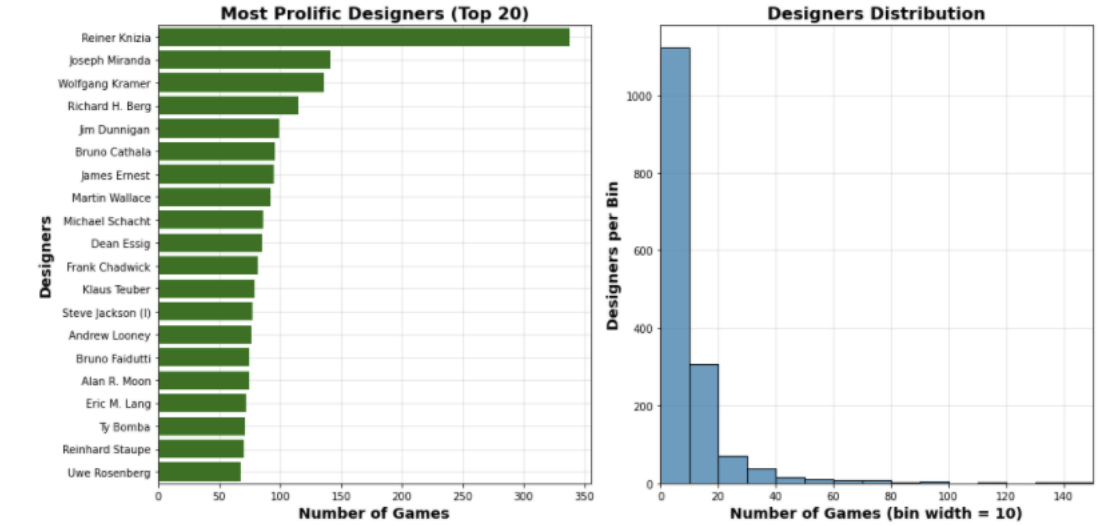
Functions 2

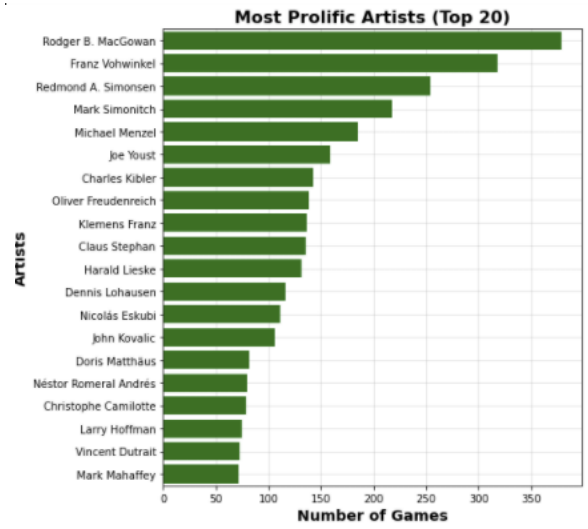
Created functions to graph the top X number of values in my features and show the distribution of those features across the category.

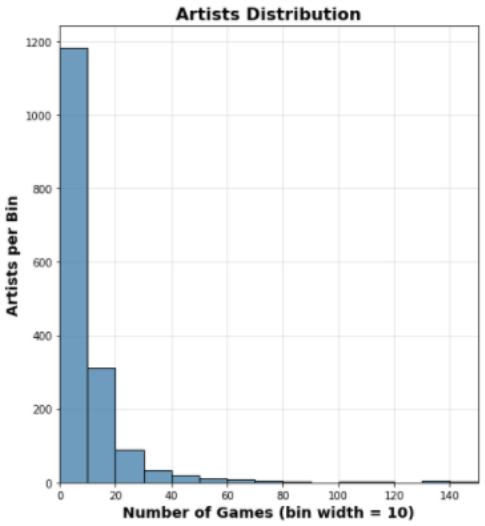
[23]
```python
# Feature Distribution Graphs
def feature_dist(data, col, topX = 20, binW = 10):

    values = data[col].value_counts()

    fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (14,7))
    sns.barplot(x = values.head(topX),y = values.head(topX).index, color = 'Green', ax = axes[0])
    axes[0].set_xlabel('Number of Games', weight = 'semibold', fontsize = 14)
    axes[0].set_ylabel(f'{col}s', weight = 'semibold', fontsize = 14)
    axes[0].set_title(f'Most Prolific {col}s (Top {topX})', weight = 'semibold', fontsize = 16)
    axes[0].set_axisbelow(True)
    axes[0].grid(alpha = .4)
    sns.histplot(values, binwidth=binW, binrange=(0,150), color = '#127ead', ax=axes[1])
    axes[1].set_xlim(0,150)
    axes[1].set_xlabel(f'Number of Games (bin width = {binW})', weight = 'semibold', fontsize = 14)
    axes[1].set_ylabel(f'{col}s per Bin', weight = 'semibold', fontsize = 14)
    axes[1].set_title(f"{col}s Distribution", weight = 'semibold', fontsize = 16)
    axes[1].set_axisbelow(True)
    plt.grid(alpha = .4)
    plt.tight_layout()
    plt.show();

# Feature Distribution Graphs 2
def feature_dist2(data, col, topX = 20):
    values = data[col].value_counts()

    fig, ax = plt.subplots(figsize = (18,6))
    sns.barplot(y = values.head(topX),x = values.head(topX).index, color = '#A90200', alpha = .9)
    plt.xlabel(f'{col}s (Top 40)', weight = 'semibold', fontsize = 14)
    plt.ylabel('Number of Games', weight = 'semibold', fontsize = 14)
    plt.title(f"Top {col}s", weight = 'semibold', fontsize = 18)
    ax.set_axisbelow(True)
    plt.grid(alpha = .4)
    plt.xticks(rotation=60, ha = "right", rotation_mode = 'anchor')
    plt.tight_layout()
    plt.show();
```

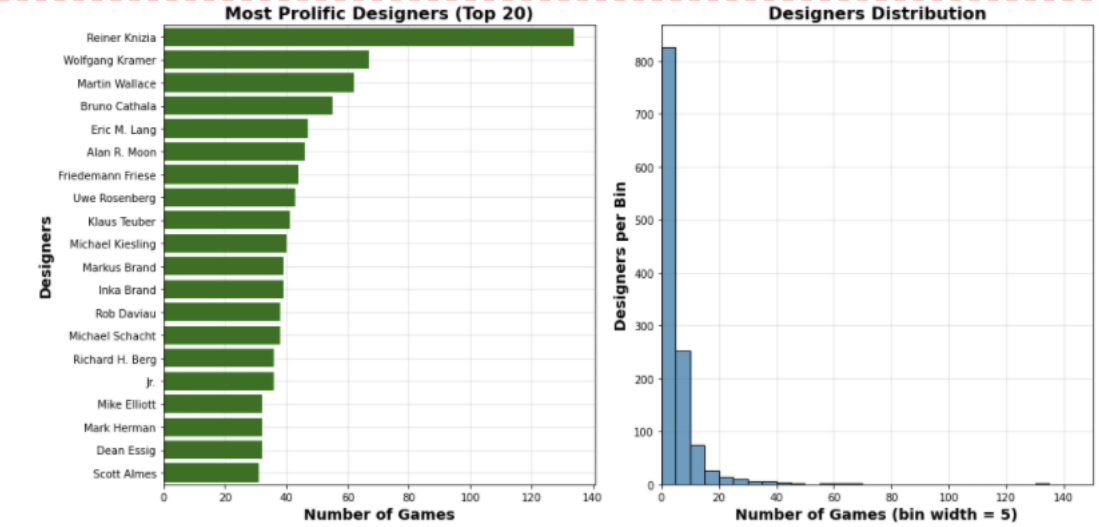
Designers and Artists

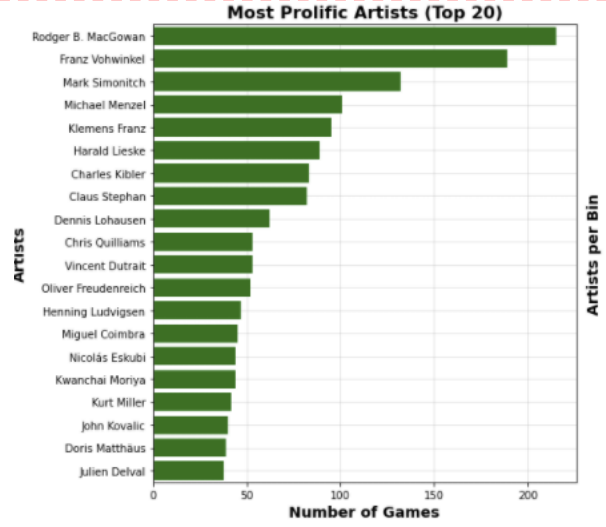
There are a number of very prolific designers and artists but based on the distribution of the data those are the exception not the rule. Note the distribution graphs cut off at 150 all but one designer and six artists.

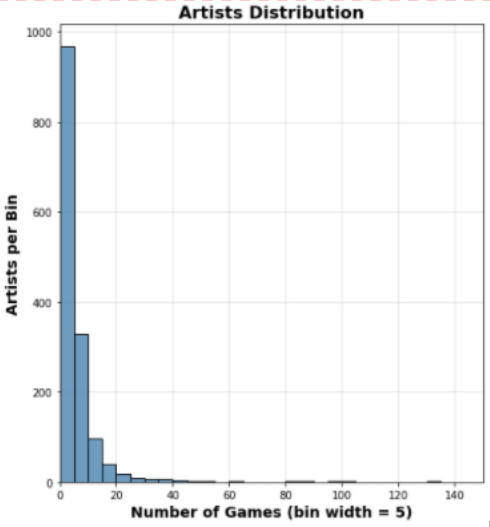






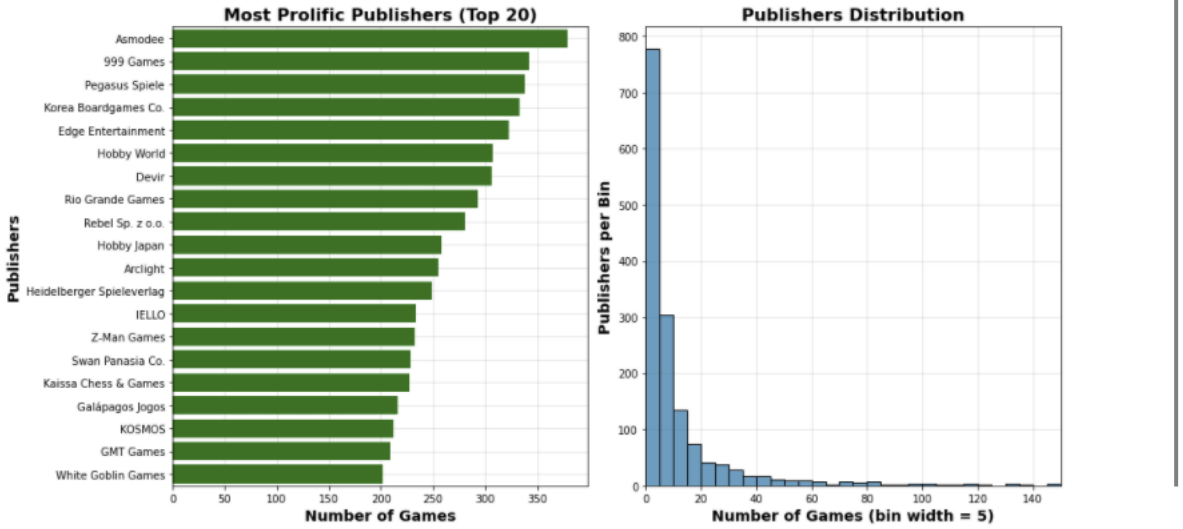






Publishers

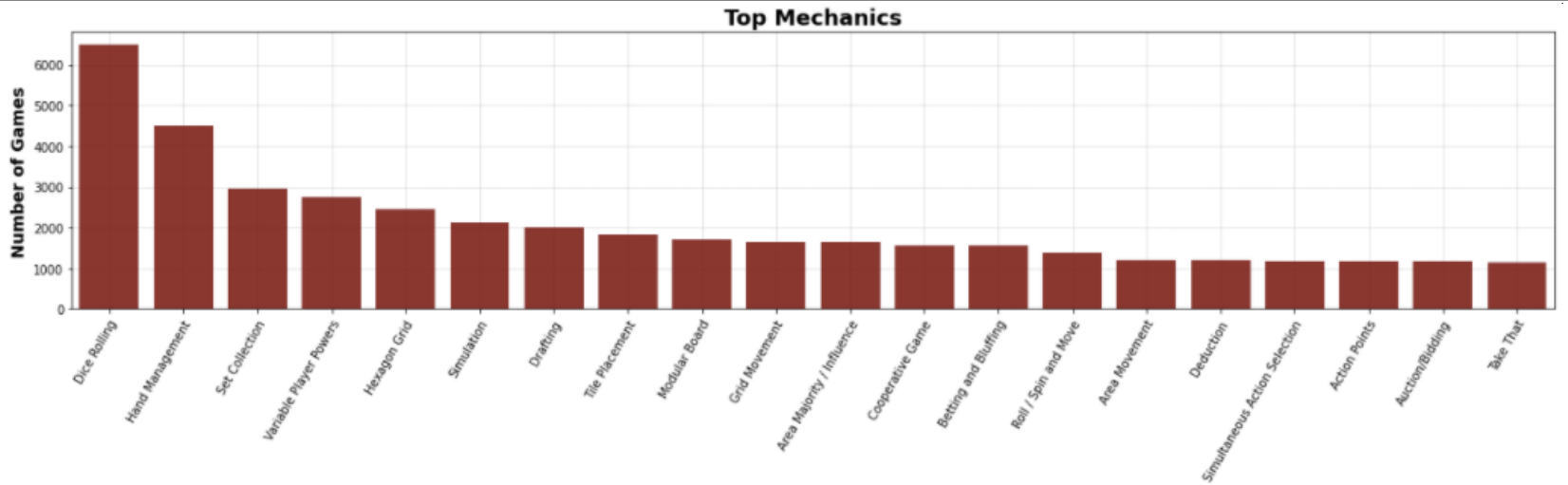
Similar results to the designer and artists graphs. Note that when you compare the entire dataset (top) to the top 5,000 (bottom) you lose some big name publishers (eg: Hasboro, Parker Brothers, Milton Bradley, etc).



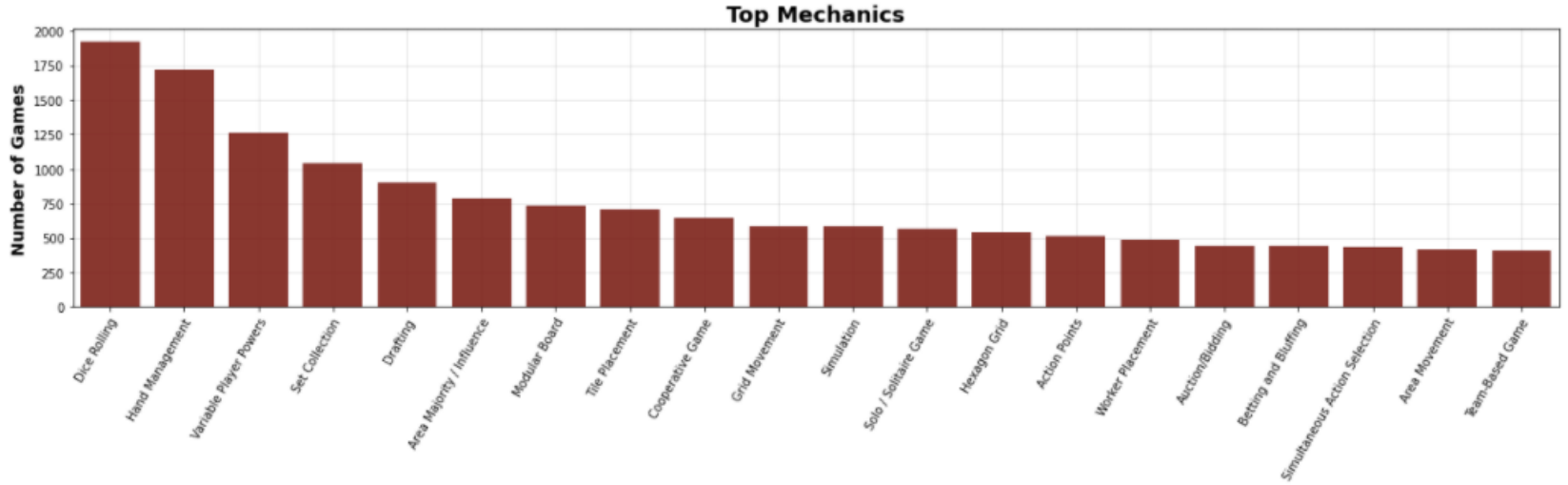
Mechanics

Of 158 possible mechanics it appears that the top 20 remain fairly consistent when compared between the entire dataset and the top 5,000. Unsurprisingly, dice rolling and hand management are highly represented.

All Database Data



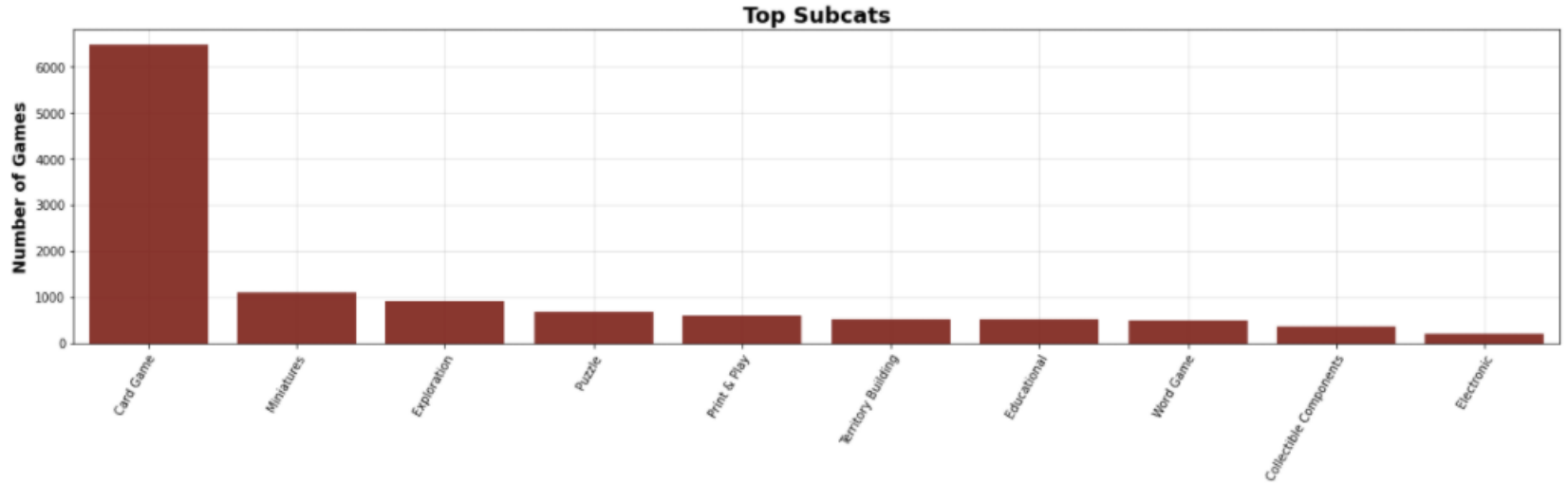
Top 5,000 Games



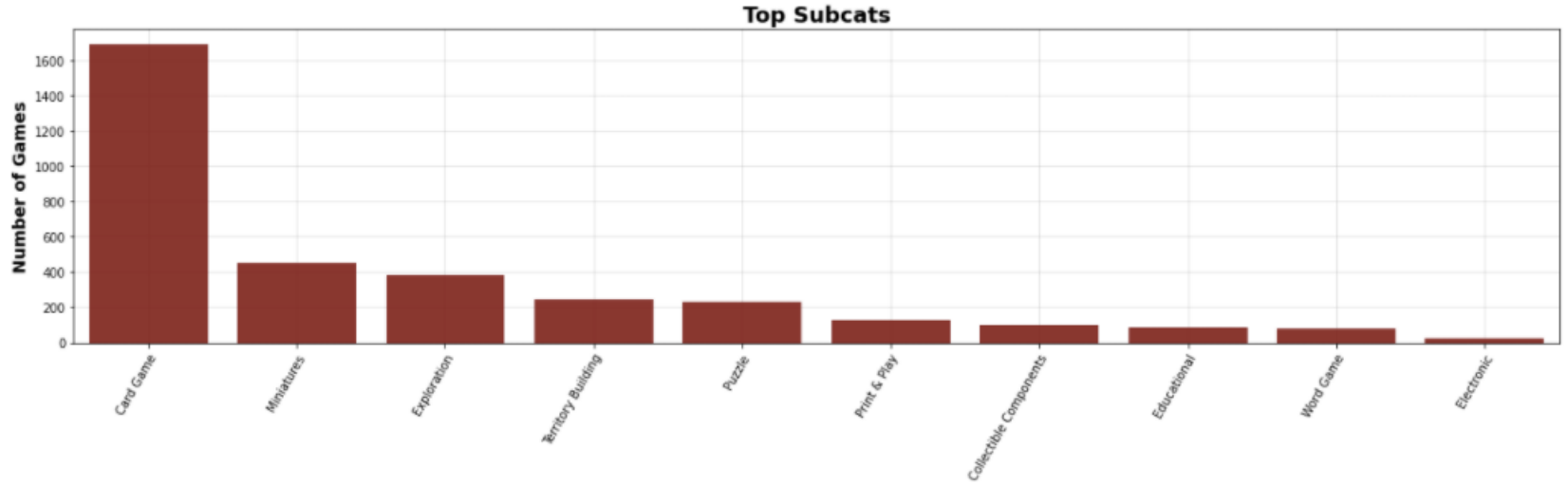
Subcategories

Card games appear to be the largest subcategory. However, it is less dominant in the top 5,000 compared to the entire dataset (4:1 vs 6:1).

All Database Data



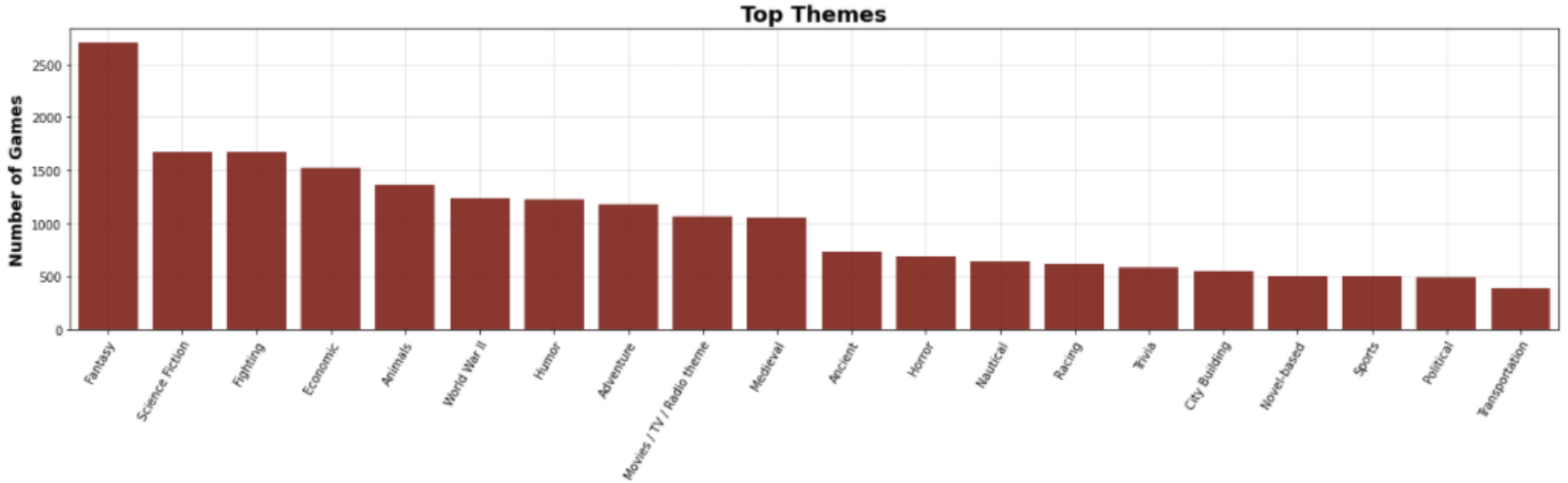
Top 5,000 Games



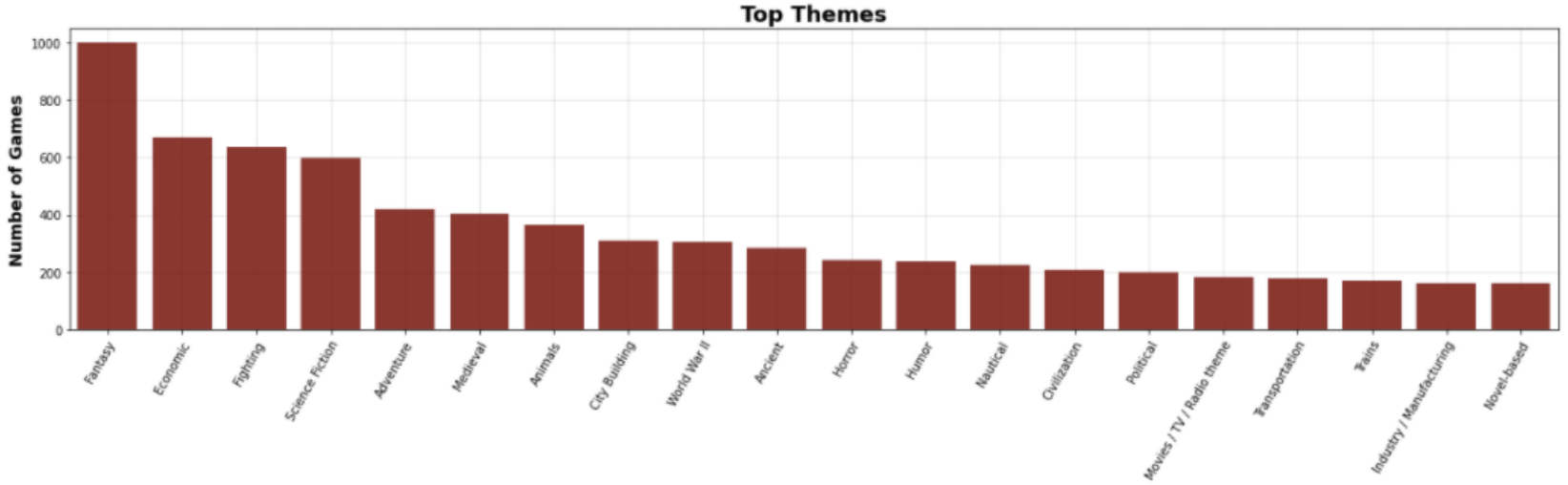
Themes

Of 218 themes listed in the database the top 20 appear to be fairly consistent with some variation within the feature. Fantasy, science fiction, fighting, economic all remain at the top (albeit with different ordering).

All Database Data



Top 5,000 Games



Merged Data

Combining the dataframes was done using pd.merge joining based on BGGId. We used how = left/right to force the dataframe down into the top 5,000 we want to look at for machine learning.

```
[55]  1 # Merge the rank and the initial game DF..
      2 main_df = pd.merge(game_df, rank_df, how = 'right', on = 'BGGId')
      3
      4 # Lose rows that are duplicated in the other DF..
      5 main_df.drop(main_df.columns[2:20], axis = 1, inplace = True)
```

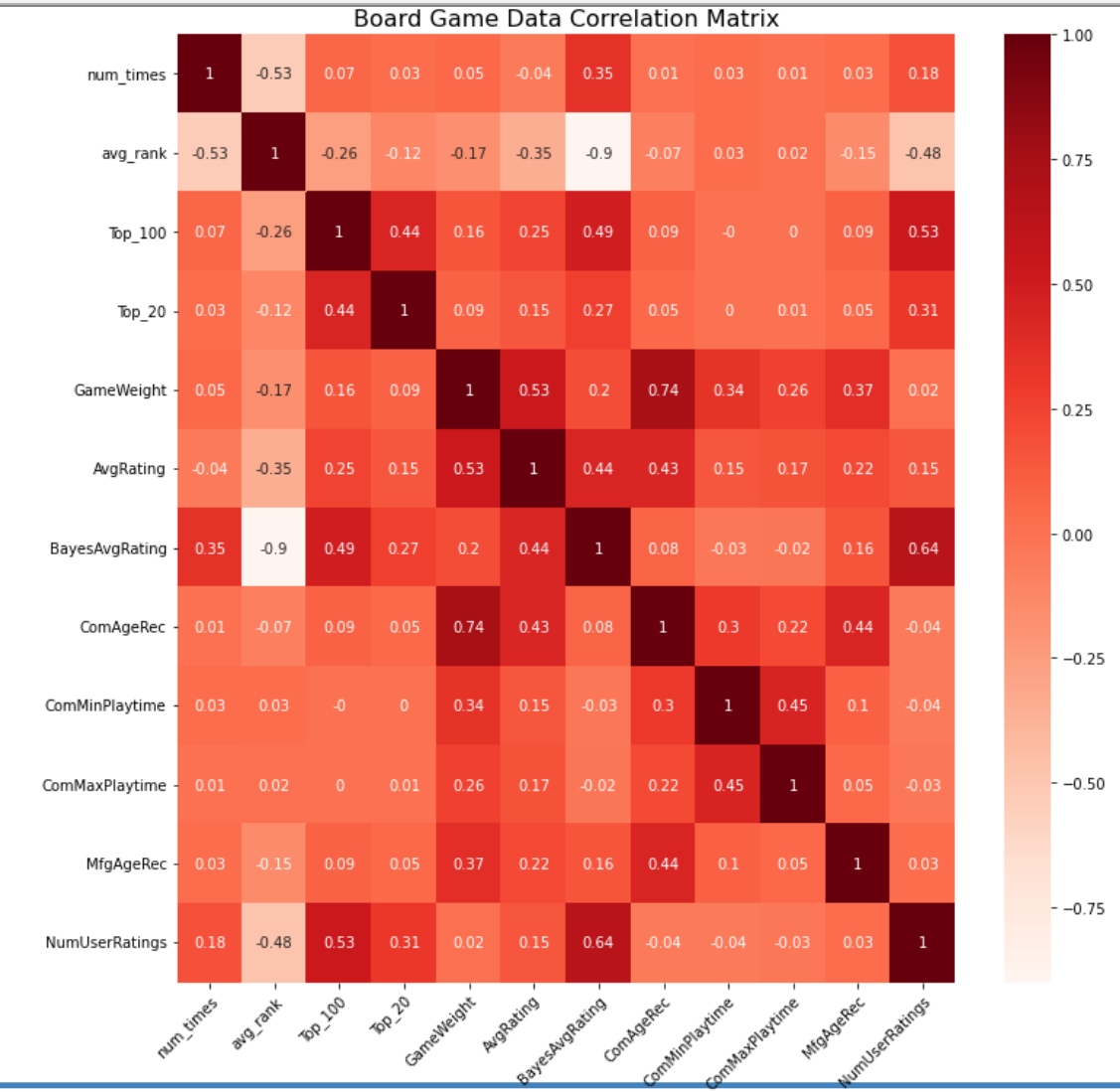
```
[56]  1 df_list = [design_df, artist_df, publish_df, mech_df, subcat_df, theme_df]
      2
      3 main2_df = games2
      4 for df in df_list:
      5     main2_df = pd.merge(main2_df, df, how = 'left', on = 'BGGId')
      6     main2_df = main2_df.drop(columns = ['ohe'])
      7
      8 main2_df.drop(columns = 'Name', inplace = True)
```

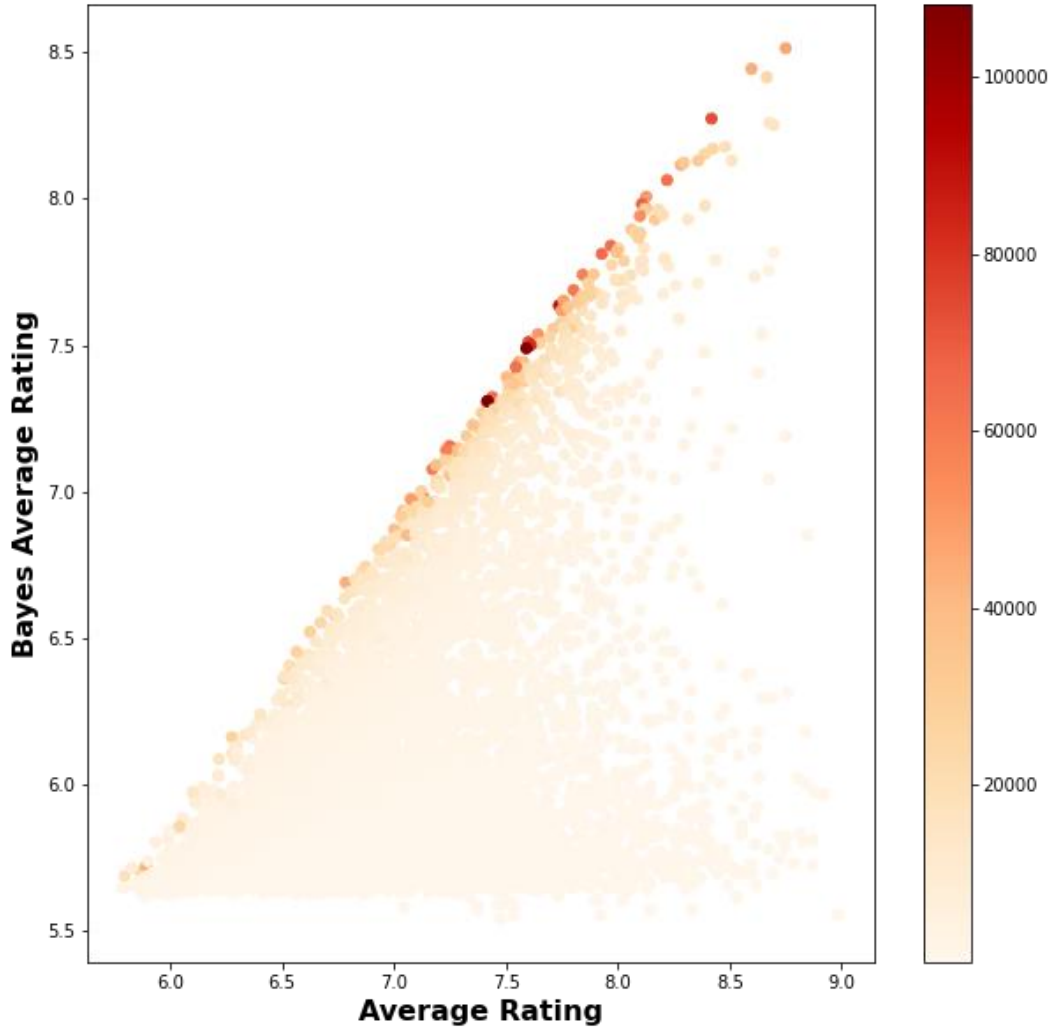
```
[57]  1 merge_main = pd.merge(main_df, main2_df, how = 'inner', on = "BGGId")
      2 merge_main.reset_index(inplace = True, drop = True)
      3 merge_main
```

Resulting Dataframe:
5909 Rows
5680 Columns

Correlations and Averages

The resulting correlation heatmap shows a large number of features with possible correlations. Some of these features are interrelated (ex: bayes avg and num of ratings) and will need to be cut depending on the ML task.





Next Steps

Feature engineering, machine learning, and feature importance

Feature Engineering:

- As noted on the previous slide some features may be duplicative or be interrelated and will need to be adjusted depending on our model. Also I would like to play around with some features to see if some work better/worse with the model (ex: community estimated playtime vs manufacturer).

Machine Learning / Feature Importance:

- Regression – I will be making regression based models (linear regression, random forest, gradient boosting) to see if our model can accurately predict BGG and/or user ratings.
- Categorization – I will be making categorization models (logistical regression, random forest, gradient boosting) to see if our model can accurately predict if a game will break into the top 100 or not.
- Feature Importance – The results will be used to see what (if any) features are important to the above.

Stretch Goal – Recommendation Engine:

- Time permitting my plan is to create a recommendation engine using the results of the ML models as a guide to what features are important. Will also probably look at clustering with user ratings as a possible means to make good recommendations.

Questions?

Prepared by: Michael McCann

Contact: msmccann10@gmail.com