By: Sheena, Laura, & Aleena

Game Boards

*NO WIFI REQUIRED

On the topic of Board Games

- Board games are Social, healthy gaming options
- Board games for all age groups, number of players, and interests.
- Compared to electronic games, board games only test the skill of the game.
- Over 20,000 games were rated by actual experiences not computer generated.

Data

- **Data Source**
  https://www.kaggle.com/andrewmvd/board-games/version/2

- **Description**

  Over 20,000 games are rated on BoardGamesGeek by users and our data was scraped into a dataset we accessed using Kaggle.com

What can we do with the data?

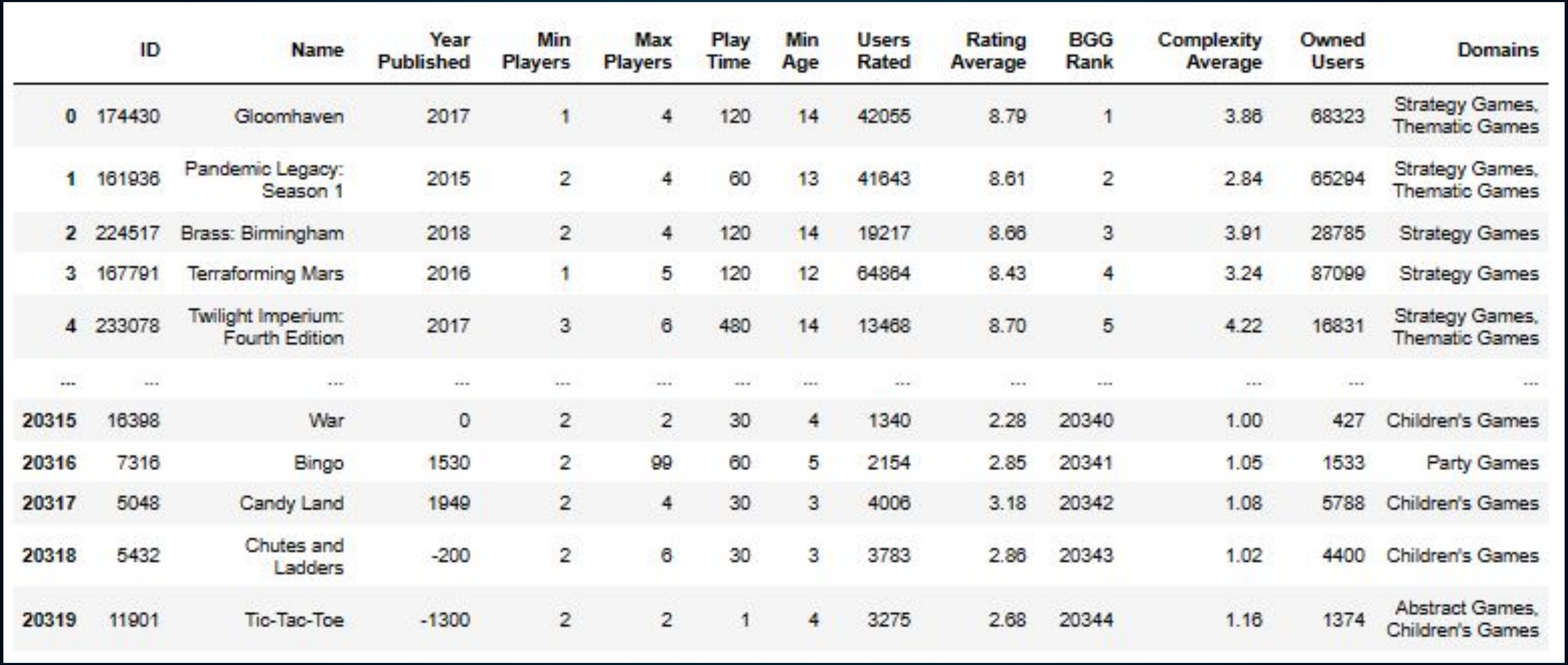
- Find correlations
- Build a model to learn what makes a game good according to BGG users.
- Complexity of games in relation to its rating average.
- Can we reasonably and accurately predict a games rating average?
- Which type of game do people like the most?

Tools

- SQL language on Postgres
- Jupyter notebook
- Sklearn libraries
- Tableau
- Visual Studio Code

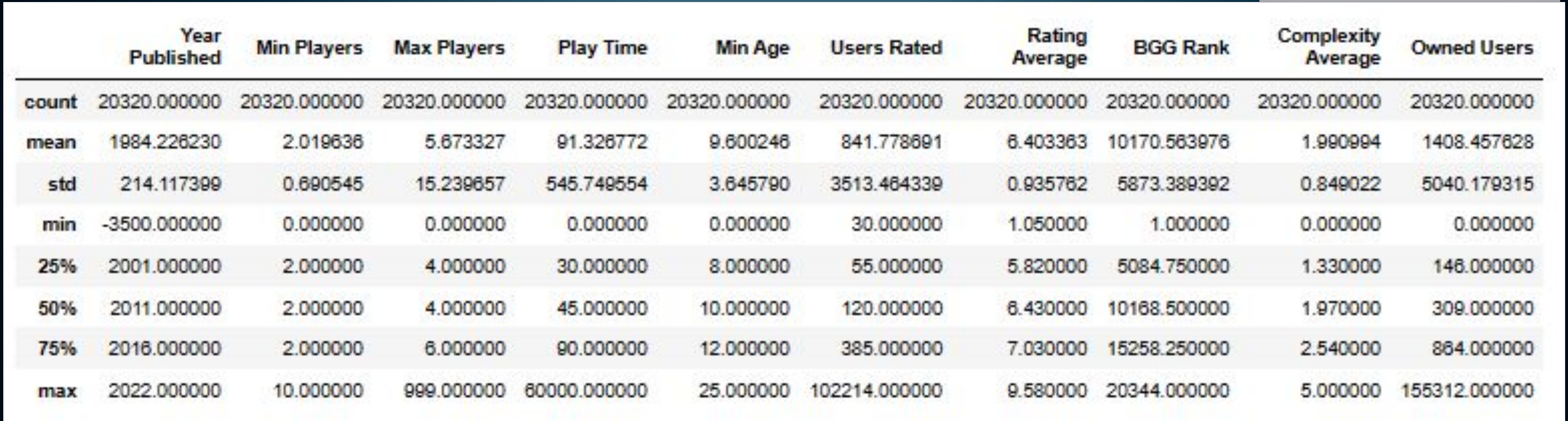
Data Exploration

Game Boards Dataframe



	ID	Name	Year Published	Min Players	Max Players	Play Time	Min Age	Users Rated	Rating Average	BGG Rank	Complexity Average	Owned Users	Domains
0	174430	Gloomhaven	2017	1	4	120	14	42055	8.79	1	3.86	68323	Strategy Games, Thematic Games
1	161936	Pandemic Legacy: Season 1	2015	2	4	60	13	41643	8.61	2	2.84	65294	Strategy Games, Thematic Games
2	224517	Brass: Birmingham	2018	2	4	120	14	19217	8.66	3	3.91	28785	Strategy Games
3	167791	Terraforming Mars	2016	1	5	120	12	64864	8.43	4	3.24	87099	Strategy Games
4	233078	Twilight Imperium: Fourth Edition	2017	3	6	480	14	13468	8.70	5	4.22	16831	Strategy Games, Thematic Games
...	...	...	...	...	...	...	...	...	...	...	...	...	...
20315	16398	War	0	2	2	30	4	1340	2.28	20340	1.00	427	Children's Games
20316	7316	Bingo	1530	2	99	60	5	2154	2.85	20341	1.05	1533	Party Games
20317	5048	Candy Land	1949	2	4	30	3	4006	3.18	20342	1.08	5788	Children's Games
20318	5432	Chutes and Ladders	-200	2	6	30	3	3783	2.86	20343	1.02	4400	Children's Games
20319	11901	Tic-Tac-Toe	-1300	2	2	1	4	3275	2.68	20344	1.16	1374	Abstract Games, Children's Games

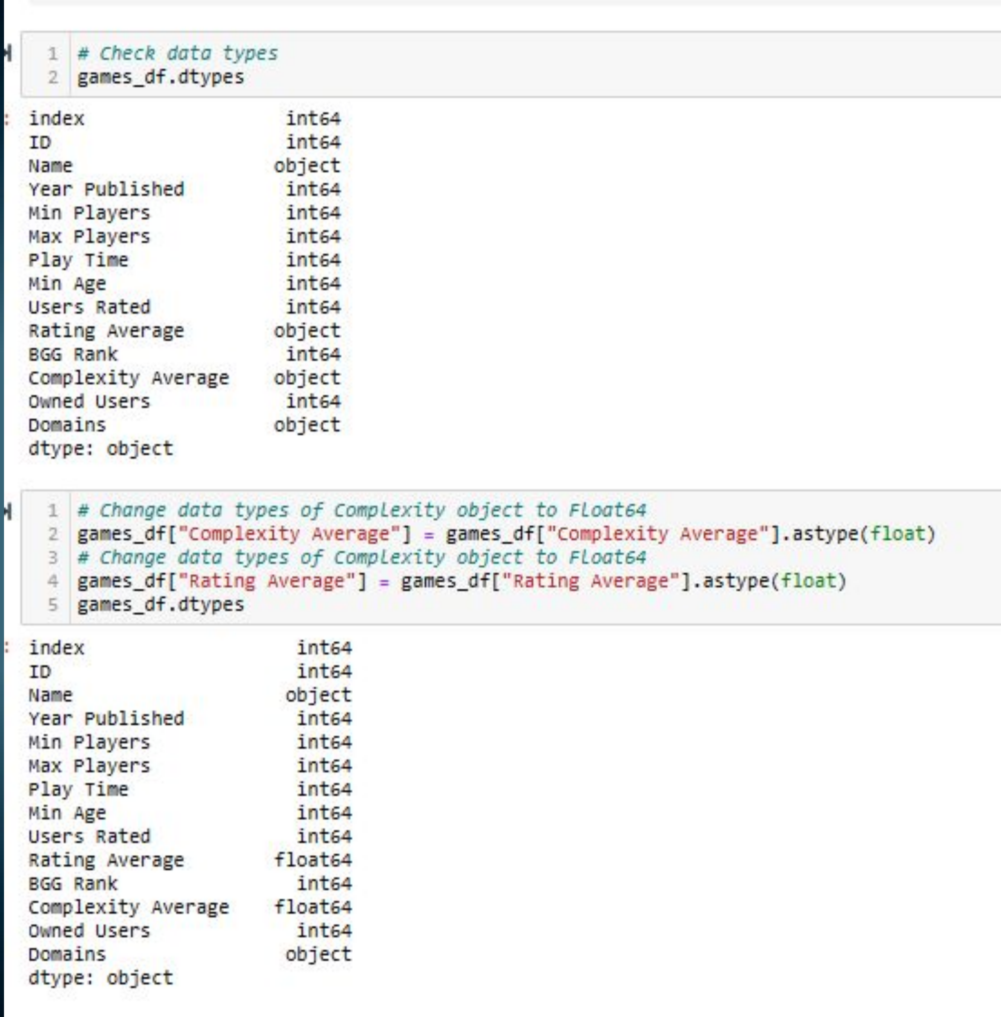
Looking at the numbers



	Year Published	Min Players	Max Players	Play Time	Min Age	Users Rated	Rating Average	BGG Rank	Complexity Average	Owned Users
count	20320.000000	20320.000000	20320.000000	20320.000000	20320.000000	20320.000000	20320.000000	20320.000000	20320.000000	20320.000000
mean	1984.226230	2.019636	5.673327	91.326772	9.600246	841.778691	6.403363	10170.563976	1.990994	1408.457628
std	214.117399	0.690545	15.239657	545.749554	3.645790	3513.464339	0.935762	5873.389392	0.849022	5040.179315
min	-3500.000000	0.000000	0.000000	0.000000	0.000000	30.000000	1.050000	1.000000	0.000000	0.000000
25%	2001.000000	2.000000	4.000000	30.000000	8.000000	55.000000	5.820000	5084.750000	1.330000	146.000000
50%	2011.000000	2.000000	4.000000	45.000000	10.000000	120.000000	6.430000	10168.500000	1.970000	309.000000
75%	2016.000000	2.000000	6.000000	90.000000	12.000000	385.000000	7.030000	15258.250000	2.540000	864.000000
max	2022.000000	10.000000	999.000000	60000.000000	25.000000	102214.000000	9.580000	20344.000000	5.000000	155312.000000

Data Types

astype() to change Rating Average and Complexity Average to float.



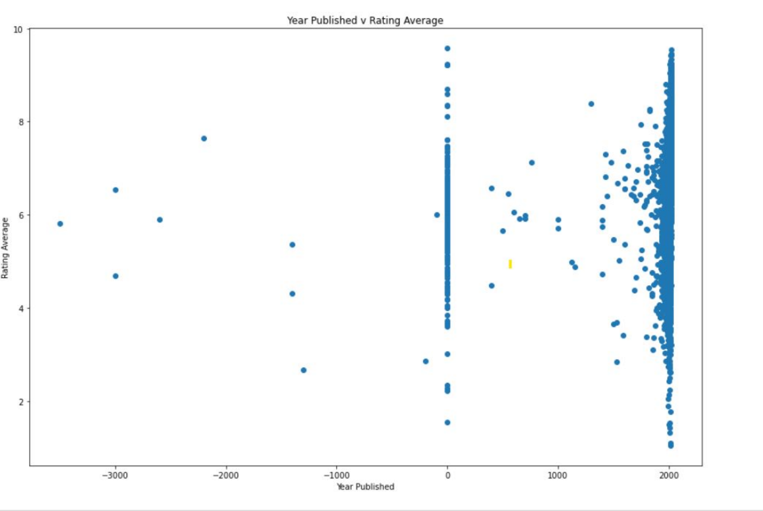
```python
# Check data types
games_df.dtypes
```

```
index                   int64
ID                      int64
Name                   object
Year Published          int64
Min Players             int64
Max Players             int64
Play Time               int64
Min Age                 int64
Users Rated             int64
Rating Average         object
BGG Rank                int64
Complexity Average     object
Owned Users             int64
Domains                object
dtype: object
```

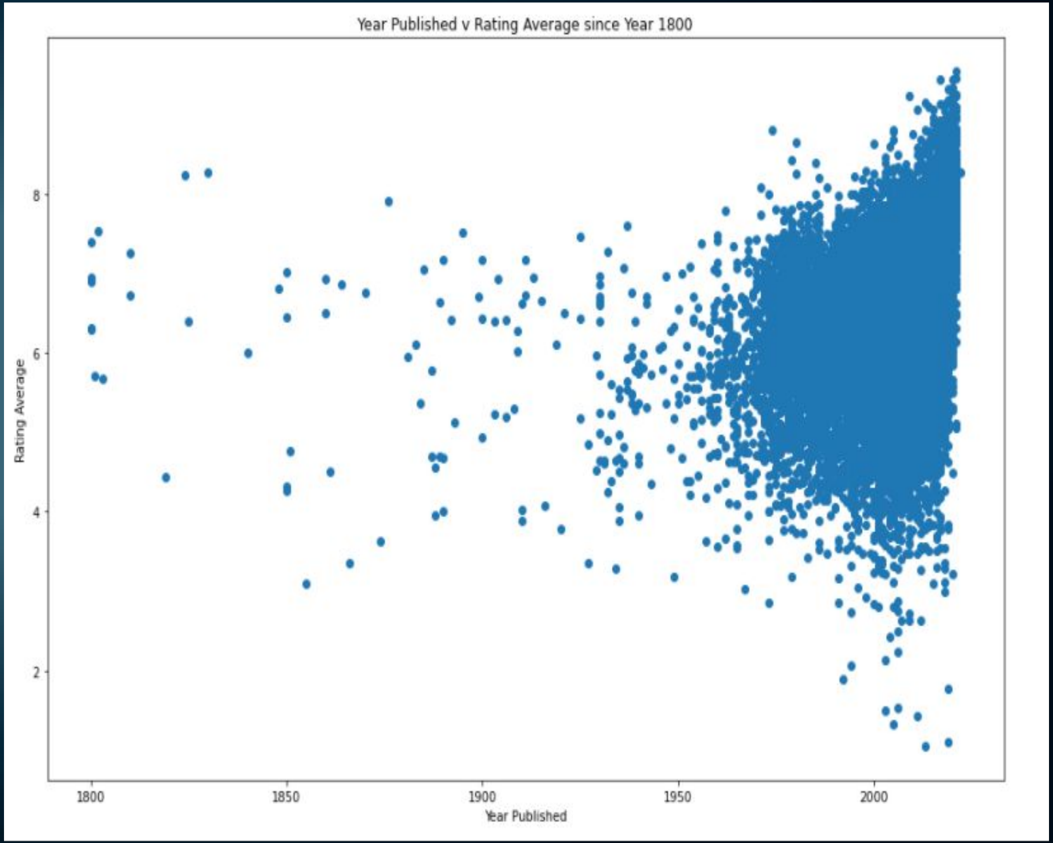
```python
# Change data types of Complexity object to Float64
games_df["Complexity Average"] = games_df["Complexity Average"].astype(float)
# Change data types of Complexity object to Float64
games_df["Rating Average"] = games_df["Rating Average"].astype(float)
games_df.dtypes
```

```
index                    int64
ID                       int64
Name                    object
Year Published           int64
Min Players              int64
Max Players              int64
Play Time                int64
Min Age                  int64
Users Rated              int64
Rating Average         float64
BGG Rank                 int64
Complexity Average     float64
Owned Users              int64
Domains                 object
dtype: object
```

Year Published v Rating Average -> Since 1800.

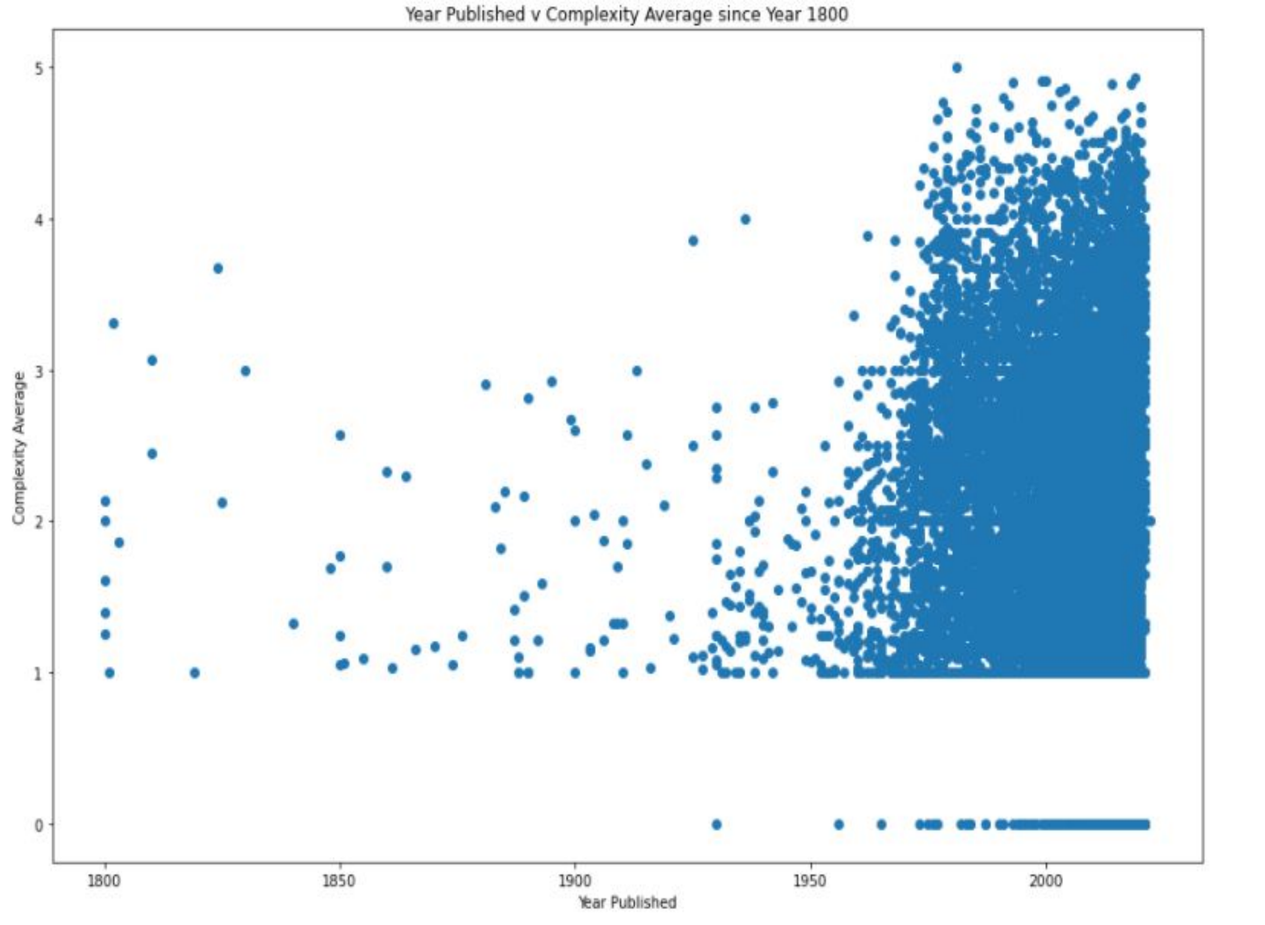


*Not filtered



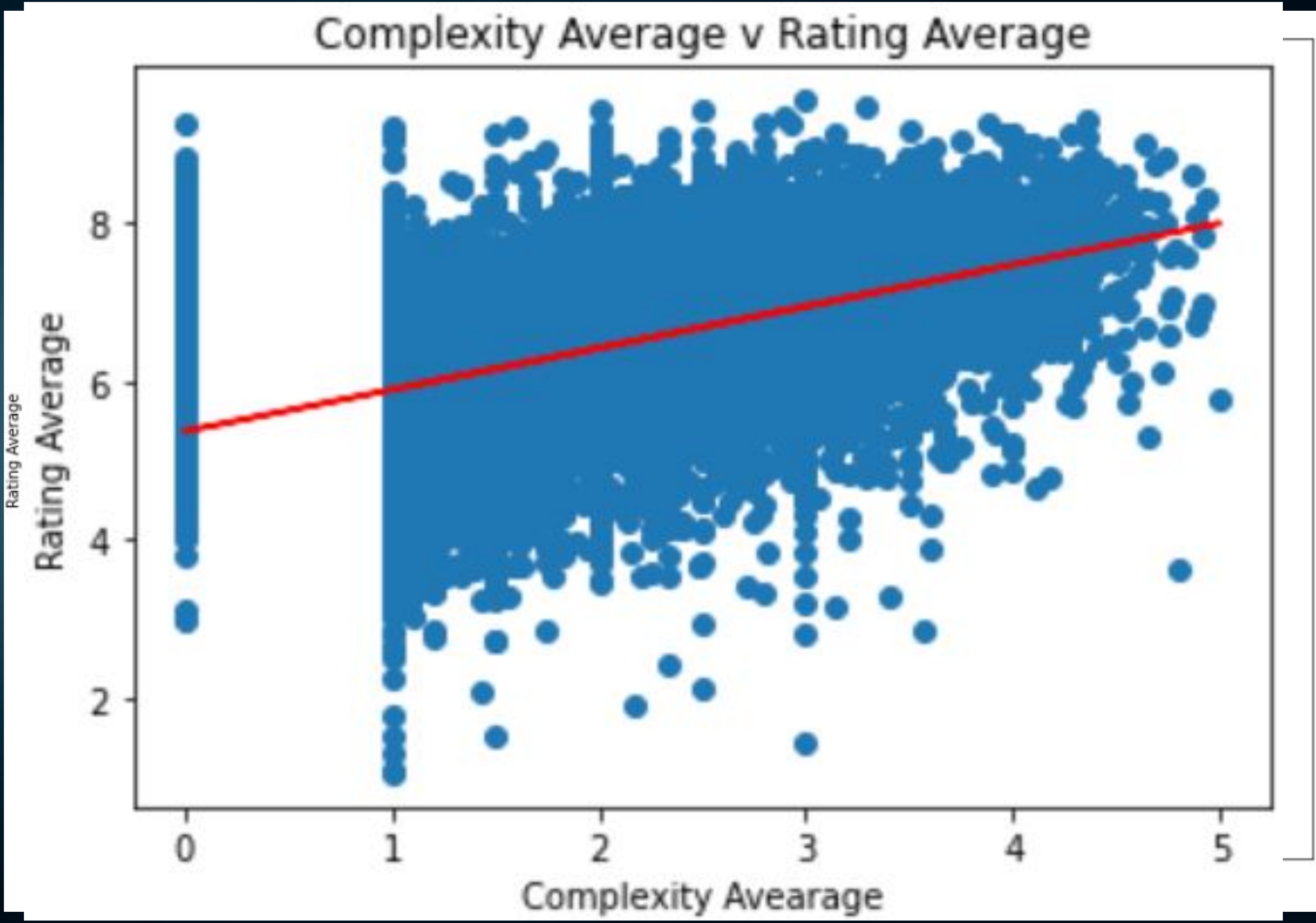
*values lower that 1800 dropped

Year Published v Complexity Average



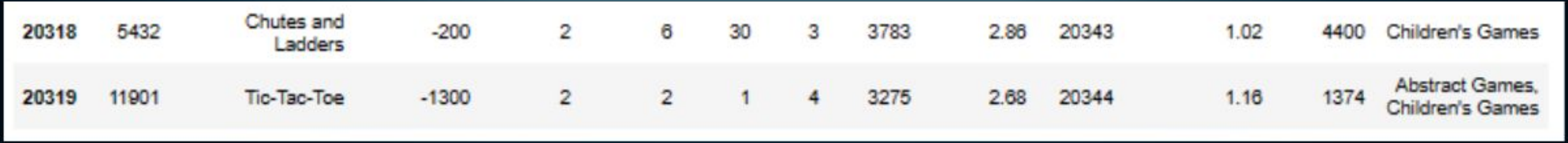
Complexity Average v Average Rating

Accuracy Score 23.8%



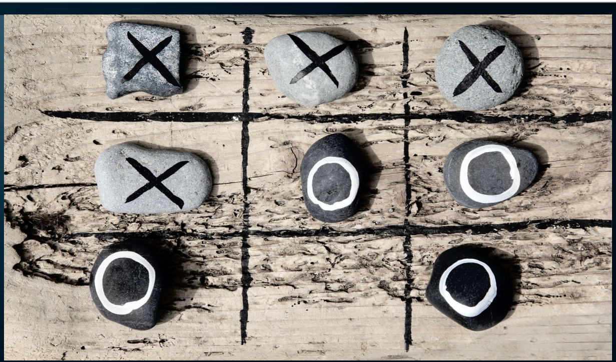
New goals:

1. All rows represented regardless of year published.



20318	5432	Chutes and Ladders	-200	2	6	30	3	3783	2.86	20343	1.02	4400	Children's Games
20319	11901	Tic-Tac-Toe	-1300	2	2	1	4	3275	2.68	20344	1.16	1374	Abstract Games, Children's Games

2. Increase accuracy score above 23.8%
3. Want to use more X features

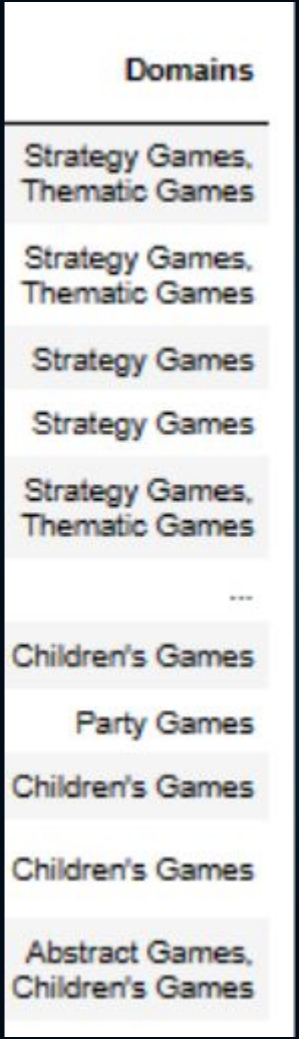


*OVERALL GOAL
Do NOT want to WASTE DATA

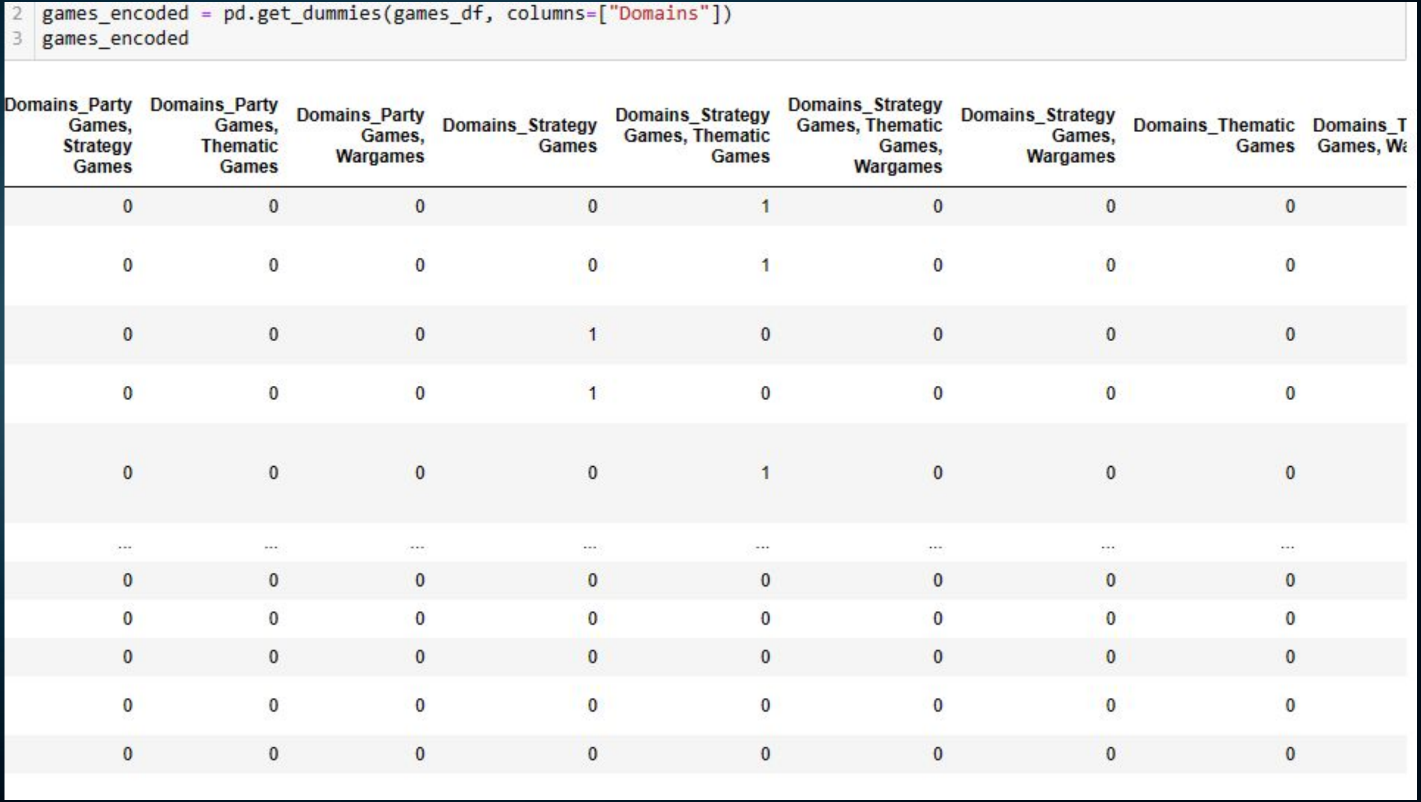
Multiple Linear Regression

* y= b0 + b1+x1 + b2x2b2 + … + bnxn

Get_dummies on Domains



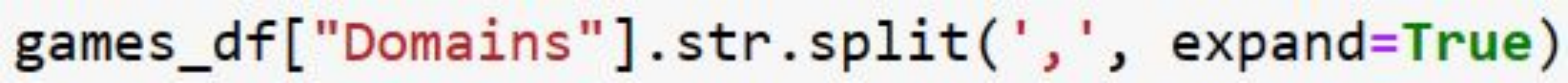
Domains
Strategy Games, Thematic Games
Strategy Games, Thematic Games
Strategy Games
Strategy Games
Strategy Games, Thematic Games
...
Children's Games
Party Games
Children's Games
Children's Games
Abstract Games, Children's Games



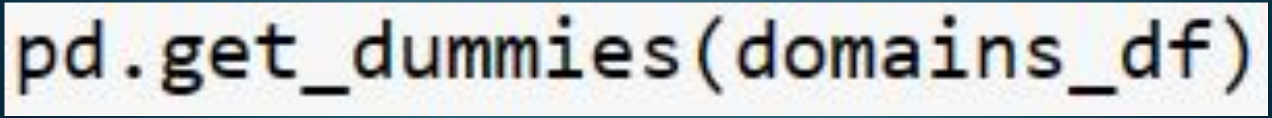
```
games_encoded = pd.get_dummies(games_df, columns=["Domains"])
games_encoded
```

Domains_Party Games, Strategy Games	Domains_Party Games, Thematic Games	Domains_Party Games, Wargames	Domains_Strategy Games	Domains_Strategy Games, Thematic Games	Domains_Strategy Games, Thematic Games, Wargames	Domains_Strategy Games, Wargames	Domains_Thematic Games	Domains_T Games, Wa
0	0	0	0	1	0	0	0	
0	0	0	0	1	0	0	0	
0	0	0	1	0	0	0	0	
0	0	0	1	0	0	0	0	
0	0	0	0	1	0	0	0	
...	...	...	...	...	...	...	...	
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	

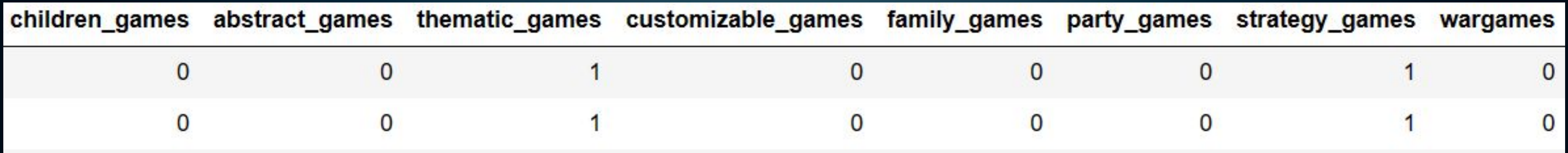
Split strings by commas, get_dummies, create new columns



```
games_df["Domains"].str.split(',', expand=True)
```

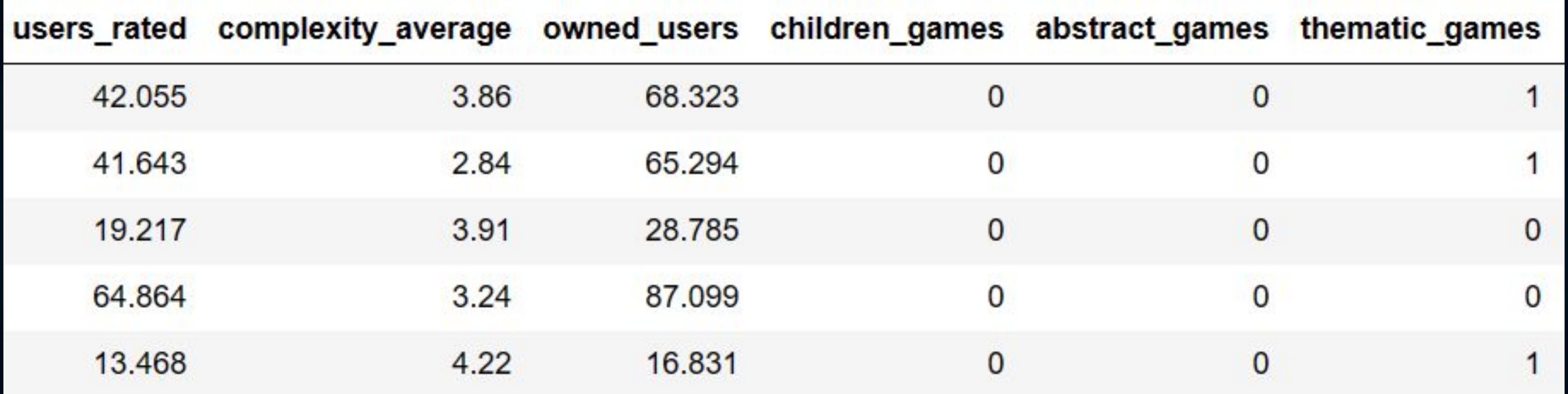


```
pd.get_dummies(domains_df)
```



children_games	abstract_games	thematic_games	customizable_games	family_games	party_games	strategy_games	wargames
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0

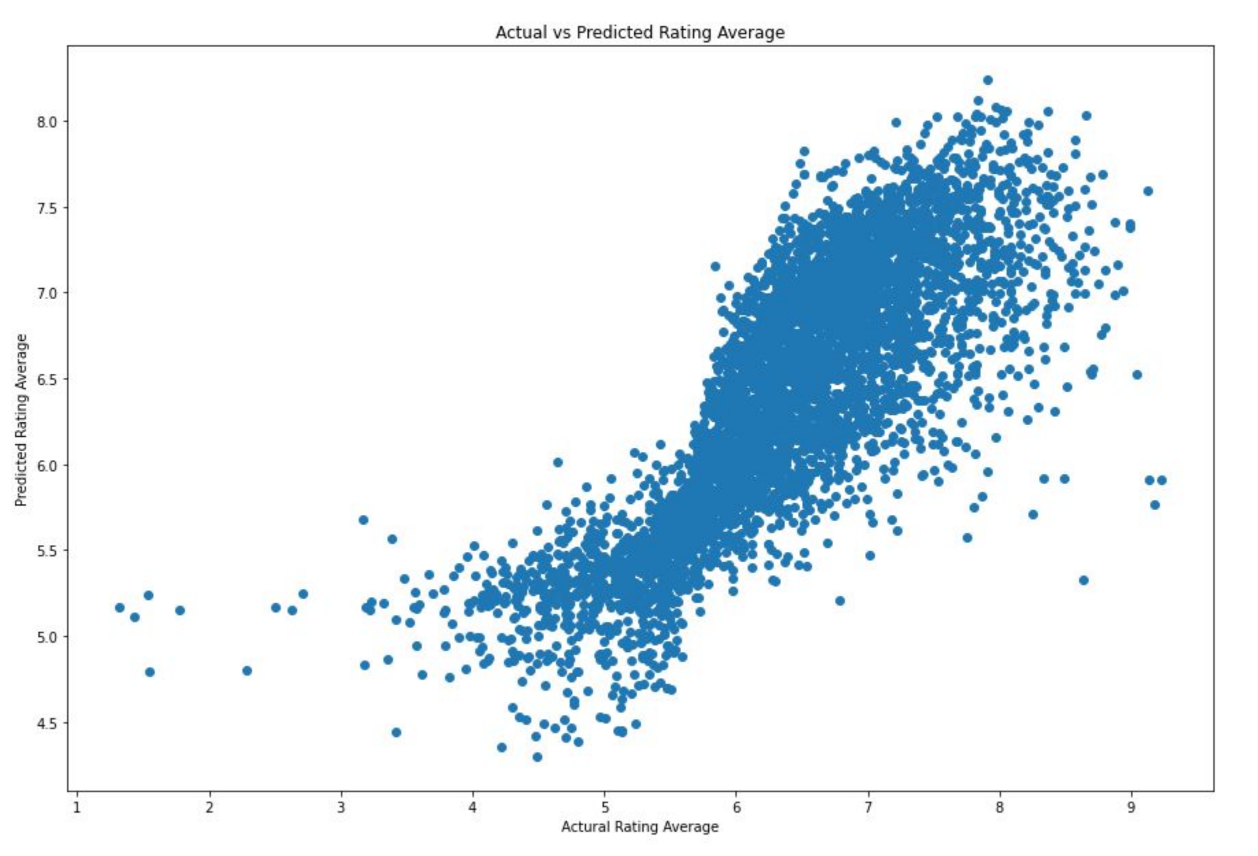
Merged and additional transformations



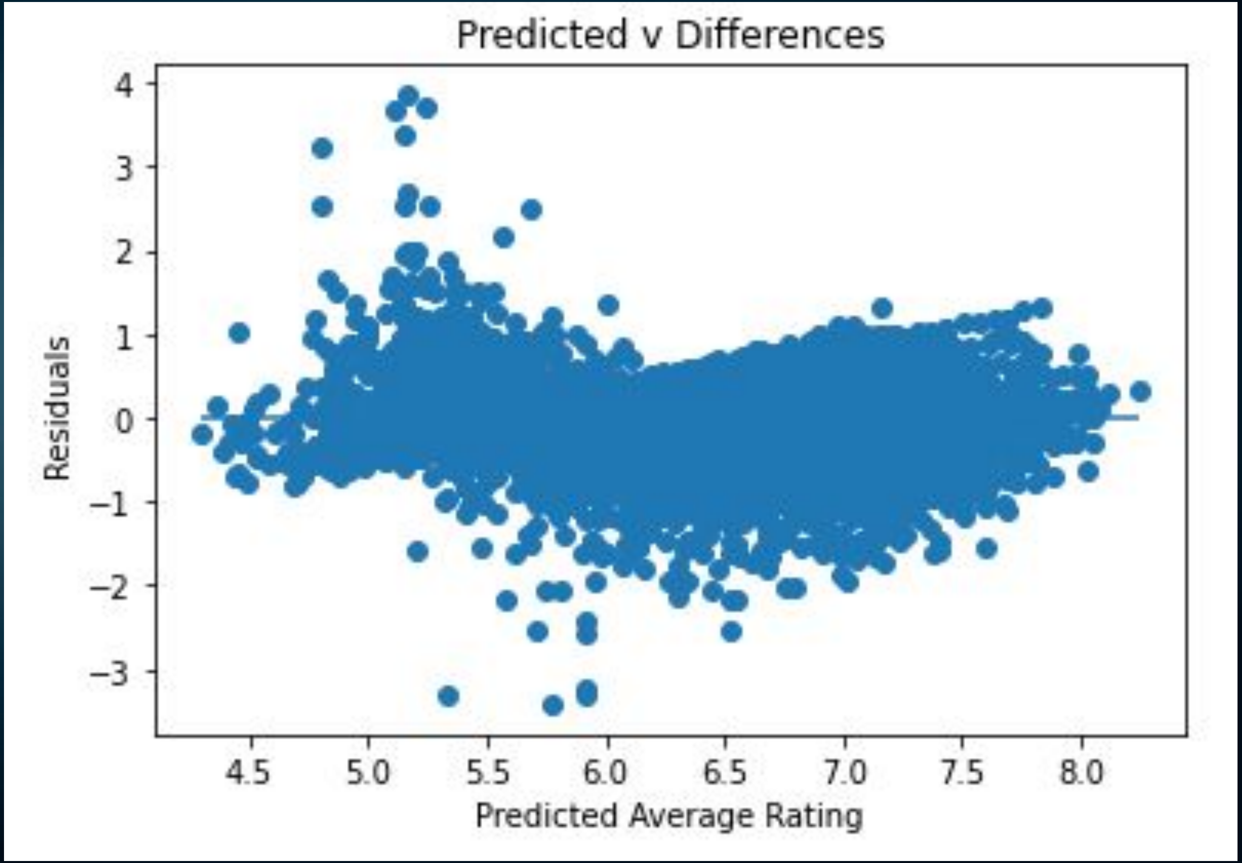
users_rated	complexity_average	owned_users	children_games	abstract_games	thematic_games
42.055	3.86	68.323	0	0	1
41.643	2.84	65.294	0	0	1
19.217	3.91	28.785	0	0	0
64.864	3.24	87.099	0	0	0
13.468	4.22	16.831	0	0	1

Multiple Linear Regression Predictions

Accuracy Score 65.6%

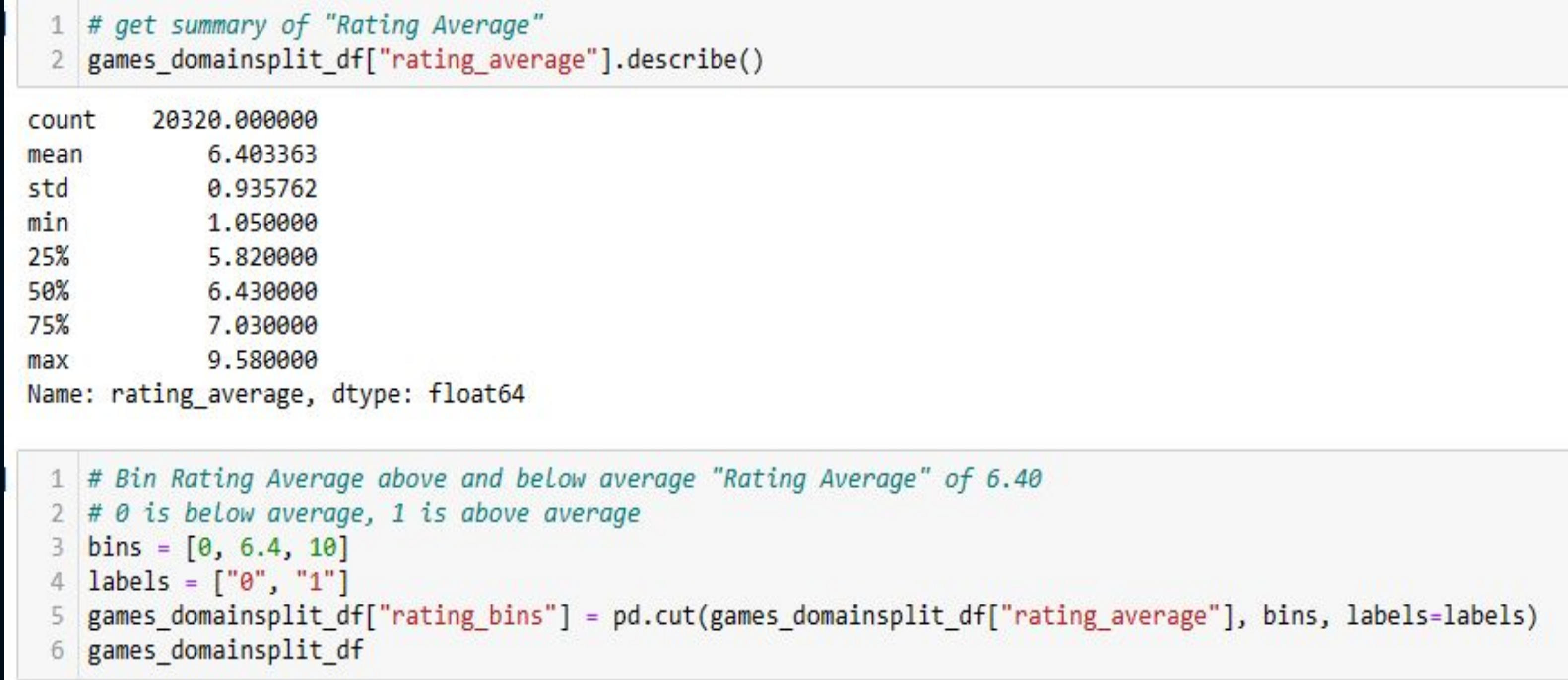


Residual Plot



Logistic Regression

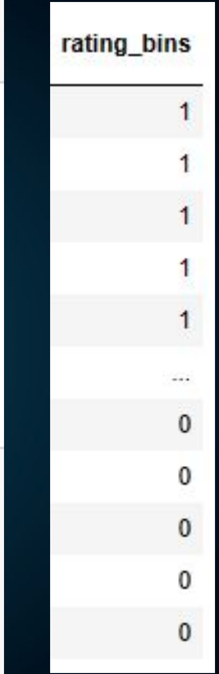
Above and Below Average



```python
# get summary of "Rating Average"
games_domainsplit_df["rating_average"].describe()
```

```
count    20320.000000
mean         6.403363
std          0.935762
min          1.050000
25%          5.820000
50%          6.430000
75%          7.030000
max          9.580000
Name: rating_average, dtype: float64
```

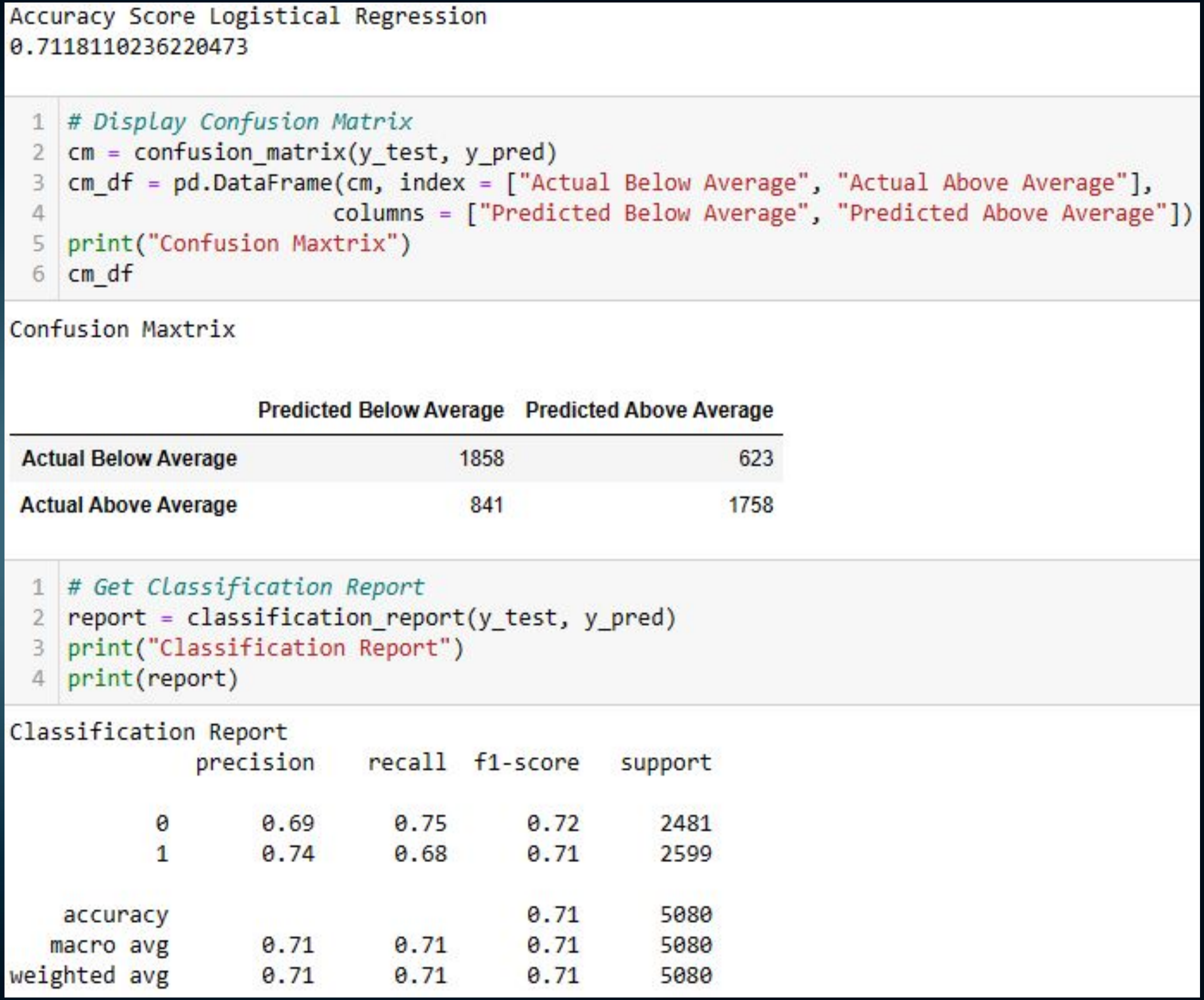
```python
# Bin Rating Average above and below average "Rating Average" of 6.40
# 0 is below average, 1 is above average
bins = [0, 6.4, 10]
labels = ["0", "1"]
games_domainsplit_df["rating_bins"] = pd.cut(games_domainsplit_df["rating_average"], bins, labels=labels)
games_domainsplit_df
```



rating_bins
1
1
1
1
1
...
0
0
0
0
0

Accuracy Score of 71.2%

- Precision of Above Average 74%



```
Accuracy Score Logistical Regression
0.7118110236220473
```

```python
# Display Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm, index = ["Actual Below Average", "Actual Above Average"],
                     columns = ["Predicted Below Average", "Predicted Above Average"])
print("Confusion Maxtrix")
cm_df
```

```
Confusion Maxtrix
```

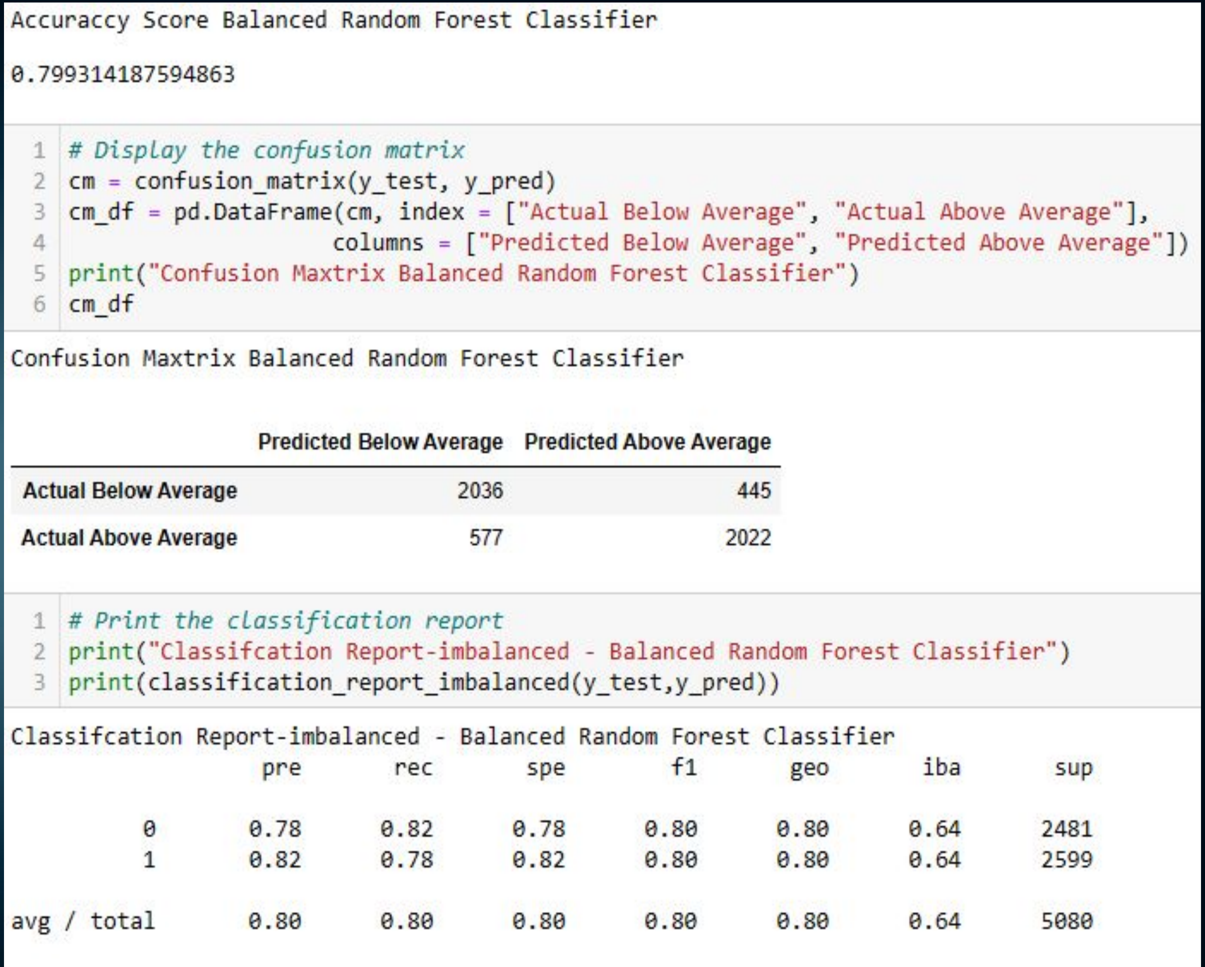
	Predicted Below Average	Predicted Above Average
Actual Below Average	1858	623
Actual Above Average	841	1758

```python
# Get Classification Report
report = classification_report(y_test, y_pred)
print("Classification Report")
print(report)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.69      0.75      0.72      2481
           1       0.74      0.68      0.71      2599

    accuracy                           0.71      5080
   macro avg       0.71      0.71      0.71      5080
weighted avg       0.71      0.71      0.71      5080
```

Accuracy Score of 79.9%

- Precision Above Average 82%



```
Accuraccy Score Balanced Random Forest Classifier

0.799314187594863
```

```
# Display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm, index = ["Actual Below Average", "Actual Above Average"],
                         columns = ["Predicted Below Average", "Predicted Above Average"])
print("Confusion Maxtrix Balanced Random Forest Classifier")
cm_df
```

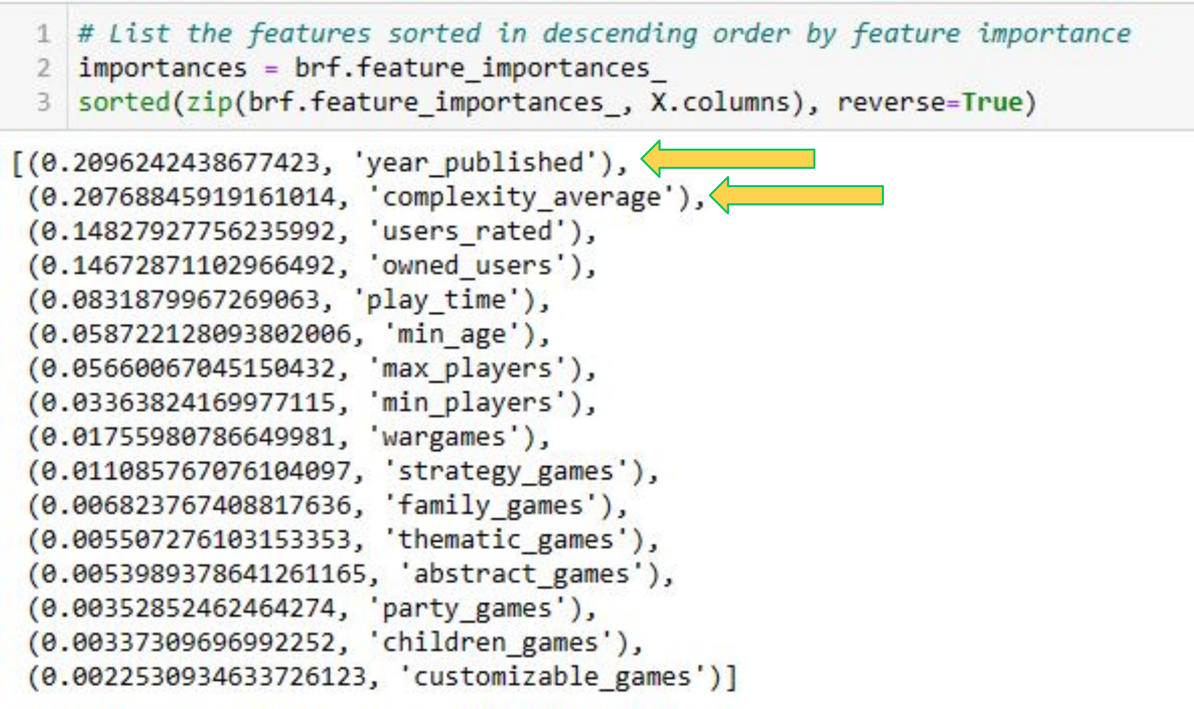
```
Confusion Maxtrix Balanced Random Forest Classifier
```

	Predicted Below Average	Predicted Above Average
Actual Below Average	2036	445
Actual Above Average	577	2022

```
# Print the classification report
print("Classifcation Report-imbalanced - Balanced Random Forest Classifier")
print(classification_report_imbalanced(y_test,y_pred))
```

```
Classifcation Report-imbalanced - Balanced Random Forest Classifier
                   pre       rec       spe        f1       geo       iba       sup

          0       0.78      0.82      0.78      0.80      0.80      0.64      2481
          1       0.82      0.78      0.82      0.80      0.80      0.64      2599

avg / total       0.80      0.80      0.80      0.80      0.80      0.64      5080
```

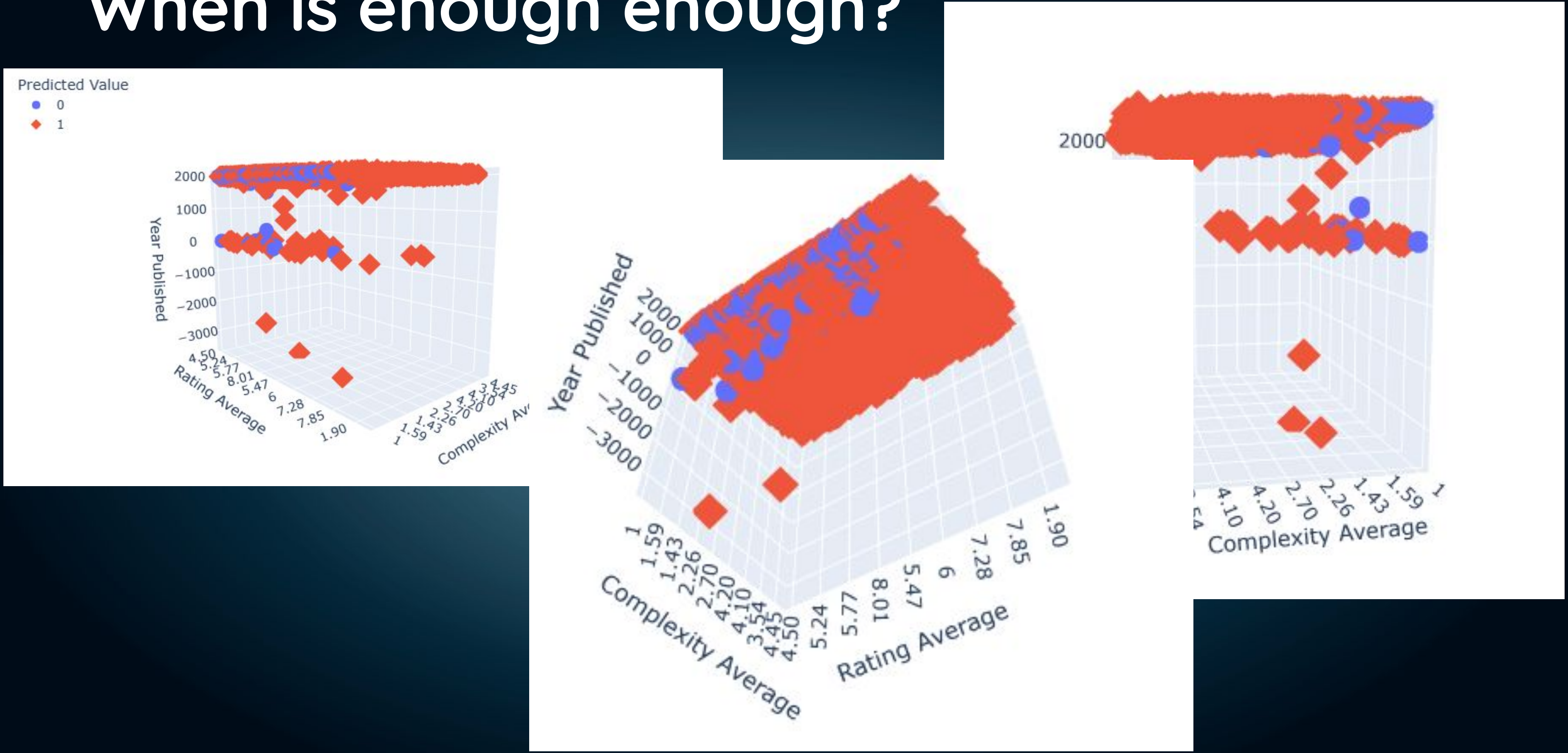
Important features

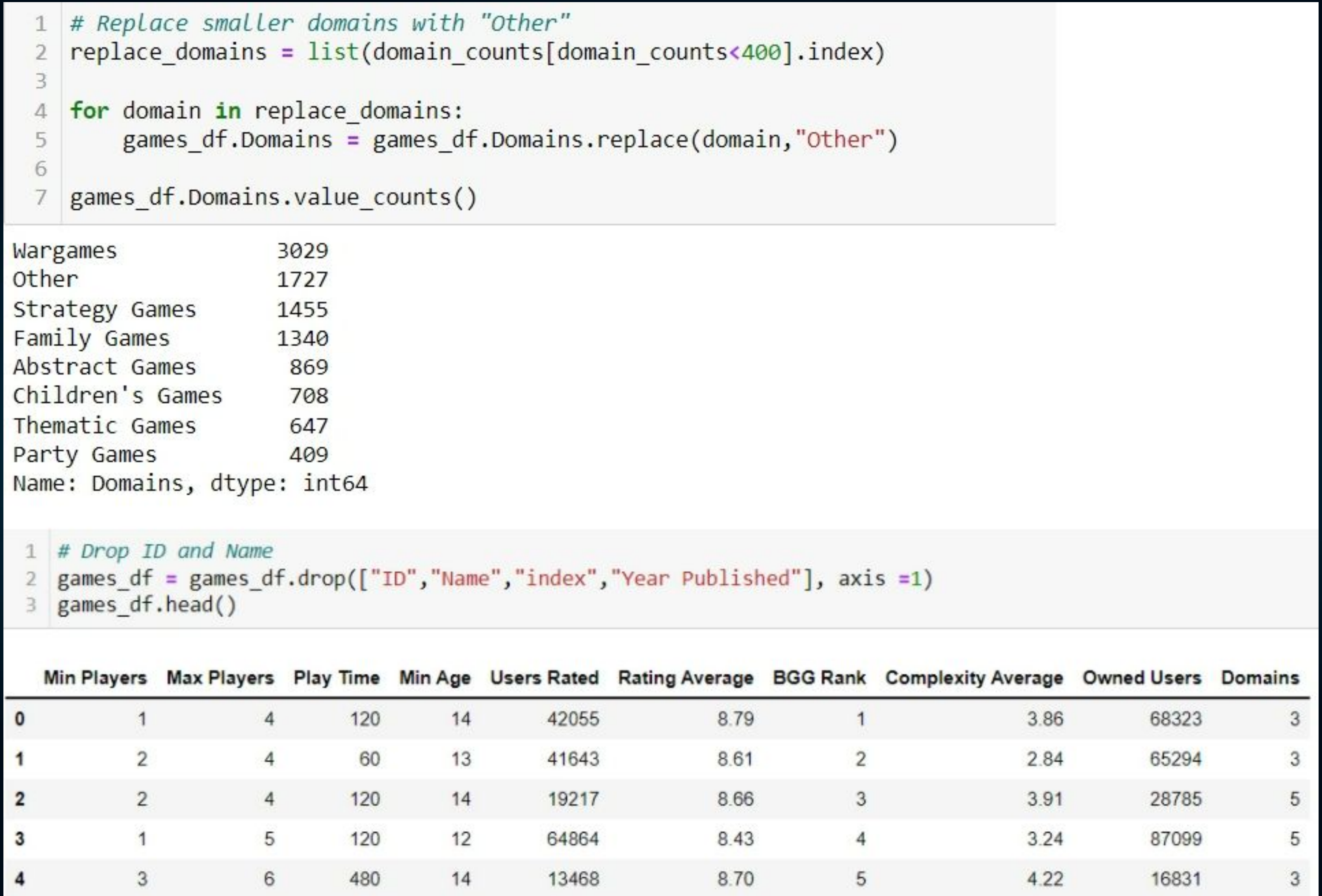


```python
# List the features sorted in descending order by feature importance
importances = brf.feature_importances_
sorted(zip(brf.feature_importances_, X.columns), reverse=True)
```

```
[(0.2096242438677423, 'year_published'),
 (0.20768845919161014, 'complexity_average'),
 (0.14827927756235992, 'users_rated'),
 (0.14672871102966492, 'owned_users'),
 (0.0831879967269063, 'play_time'),
 (0.058722128093802006, 'min_age'),
 (0.05660067045150432, 'max_players'),
 (0.03363824169977115, 'min_players'),
 (0.01755980786649981, 'wargames'),
 (0.011085767076104097, 'strategy_games'),
 (0.006823767408817636, 'family_games'),
 (0.005507276103153353, 'thematic_games'),
 (0.0053989378641261165, 'abstract_games'),
 (0.00352852462464274, 'party_games'),
 (0.00337309696992252, 'children_games'),
 (0.0022530934633726123, 'customizable_games')]
```

When is enough enough?





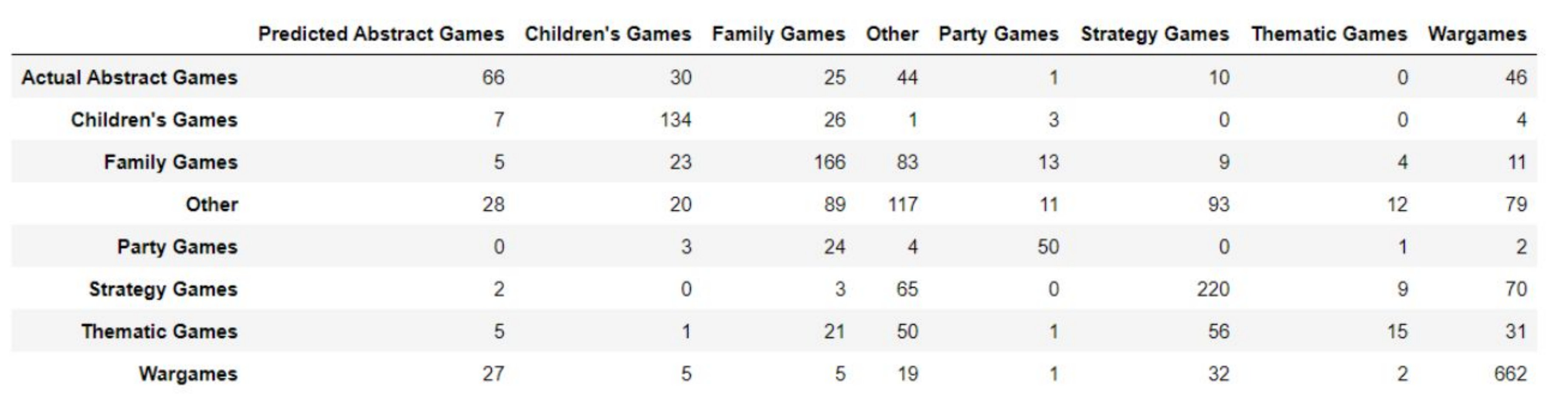
```python
# Replace smaller domains with "Other"
replace_domains = list(domain_counts[domain_counts<400].index)

for domain in replace_domains:
    games_df.Domains = games_df.Domains.replace(domain,"Other")

games_df.Domains.value_counts()
```

```
Wargames            3029
Other               1727
Strategy Games      1455
Family Games        1340
Abstract Games       869
Children's Games     708
Thematic Games       647
Party Games          409
Name: Domains, dtype: int64
```

```python
# Drop ID and Name
games_df = games_df.drop(["ID","Name","index","Year Published"], axis =1)
games_df.head()
```

	Min Players	Max Players	Play Time	Min Age	Users Rated	Rating Average	BGG Rank	Complexity Average	Owned Users	Domains
0	1	4	120	14	42055	8.79	1	3.86	68323	3
1	2	4	60	13	41643	8.61	2	2.84	65294	3
2	2	4	120	14	19217	8.66	3	3.91	28785	5
3	1	5	120	12	64864	8.43	4	3.24	87099	5
4	3	6	480	14	13468	8.70	5	4.22	16831	3

Confusion Matrix



	Predicted Abstract Games	Children's Games	Family Games	Other	Party Games	Strategy Games	Thematic Games	Wargames
Actual Abstract Games	66	30	25	44	1	10	0	46
Children's Games	7	134	26	1	3	0	0	4
Family Games	5	23	166	83	13	9	4	11
Other	28	20	89	117	11	93	12	79
Party Games	0	3	24	4	50	0	1	2
Strategy Games	2	0	3	65	0	220	9	70
Thematic Games	5	1	21	50	1	56	15	31
Wargames	27	5	5	19	1	32	2	662

Dashboard

**Tableau*

Visualizations Link

Click Here

Conclusion

- Domain (game category) cannot be reasonably predicted using other game variables
- The higher the rating average is, the higher the complexity average.
- Unable to predict rating average itself
- Average rating data is useful and important for board game manufacturers.
- Improve their board games/make more board game that are better