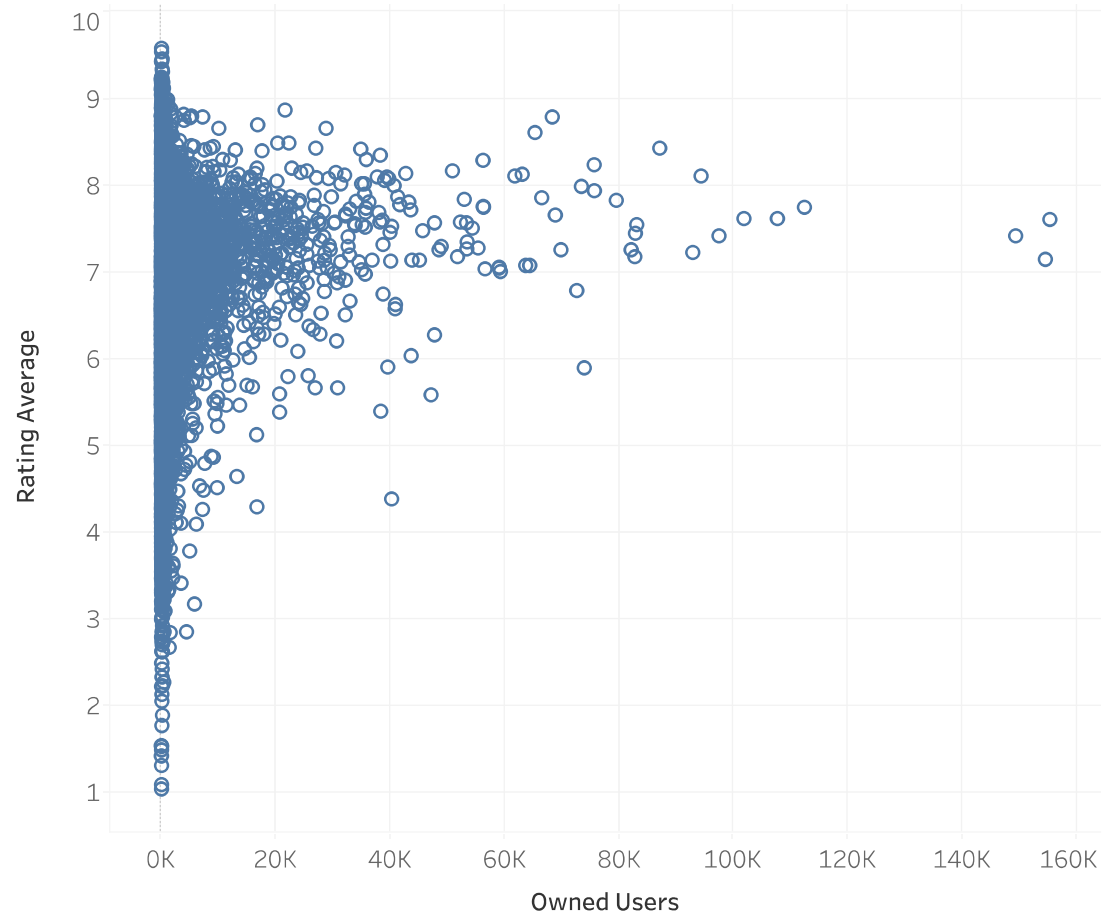
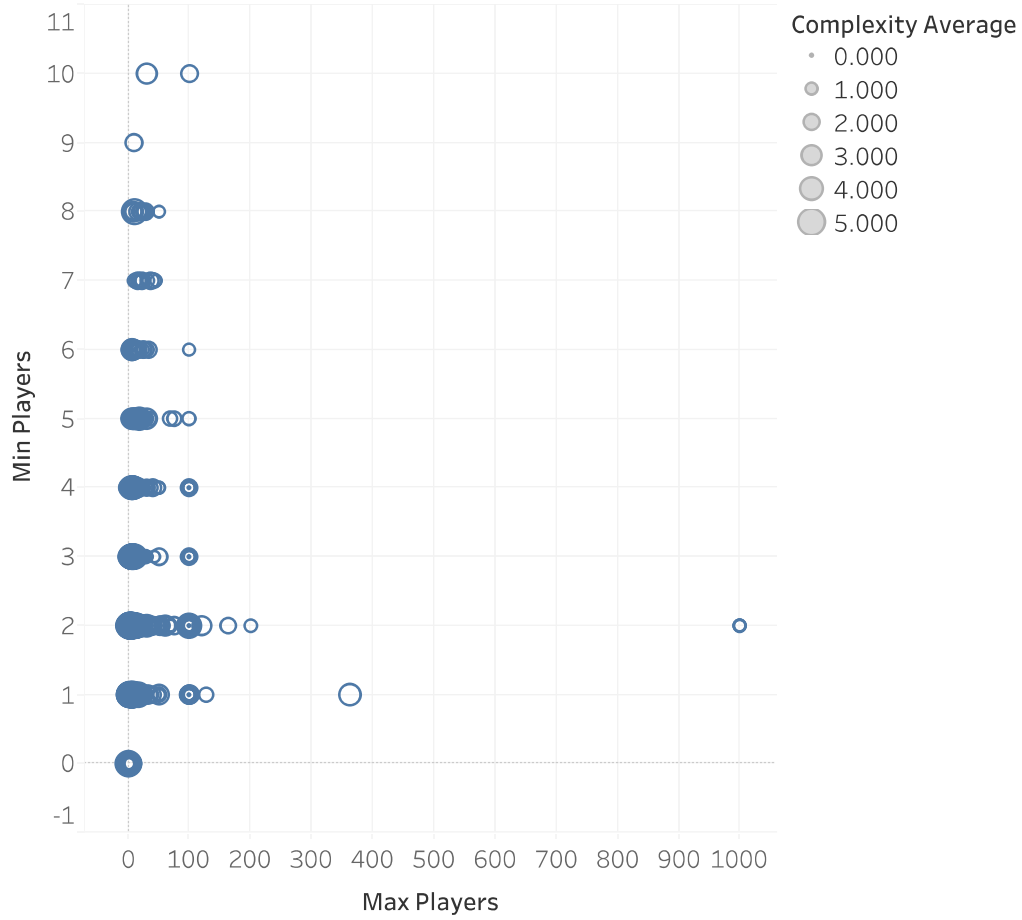


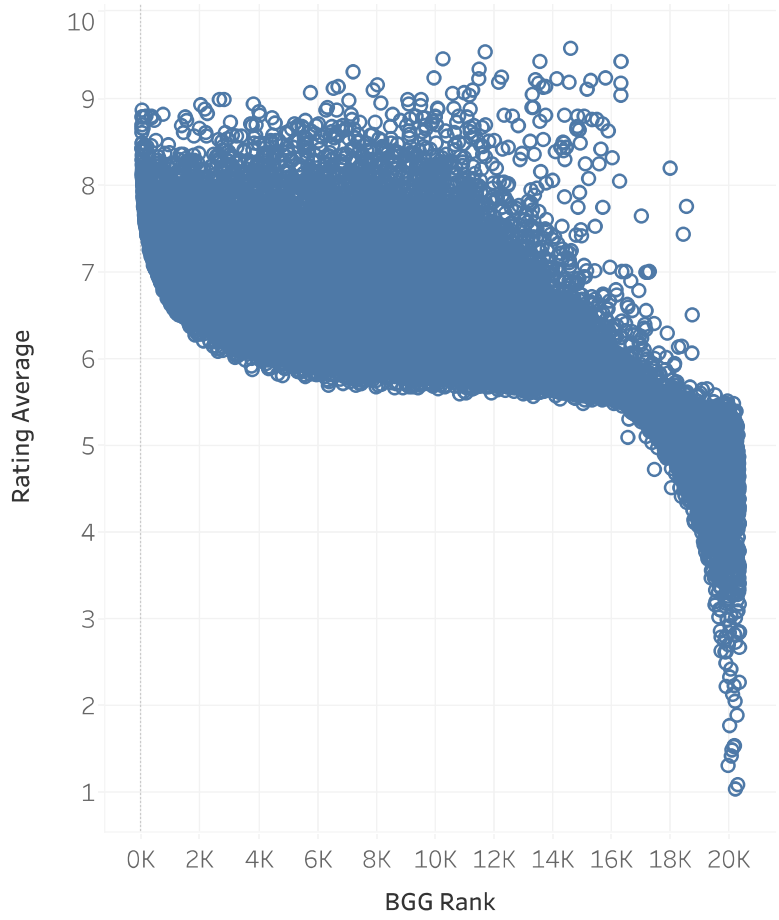
Owned Users vs. Rating Average.

Min-Max players vs Complexity Avg



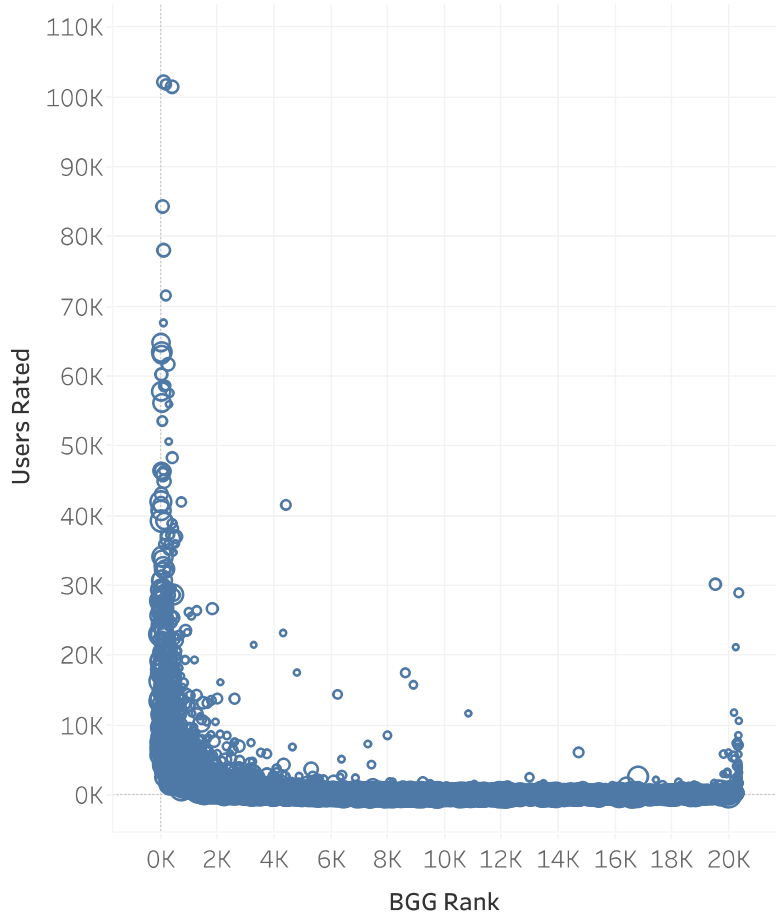
Max Players vs. Min Players. Size shows details about Complexity Average.

Rank vs Rating Avg



BGG Rank vs. Rating Average. The data is filtered on Domains (group), which keeps 9 of 9 members.

Rank vs Users Rated

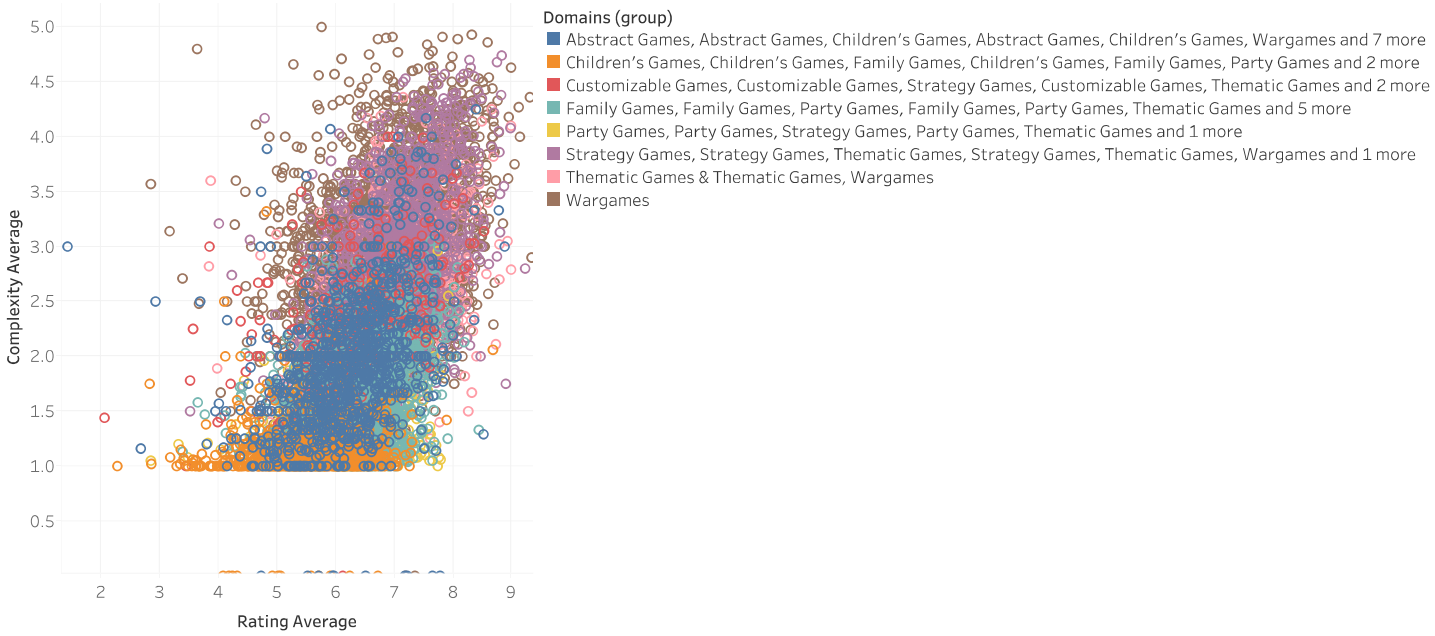


BGG Rank vs. Users Rated. Size shows details about Complexity Average.

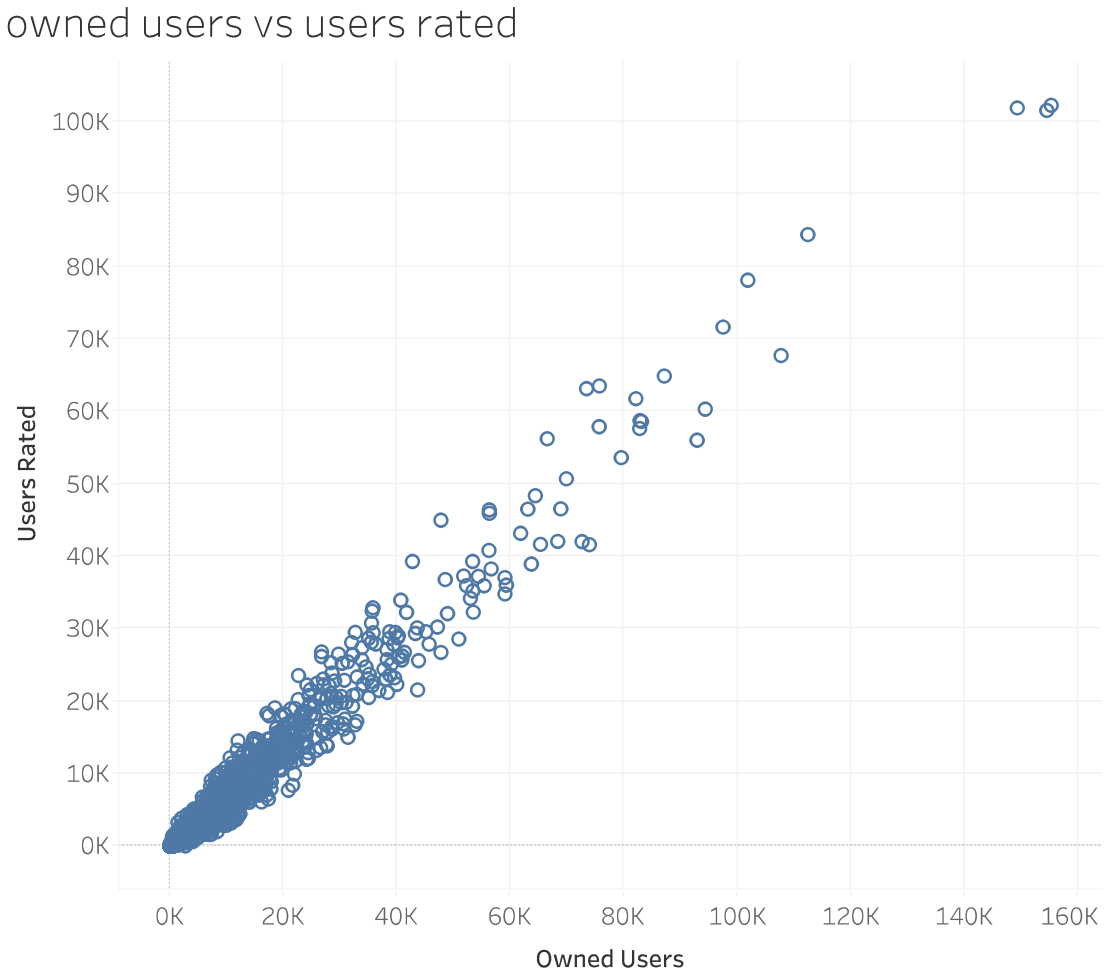
Complexity Average

- 0
- 1
- 1.01
- 1.02
- 1.03
- 1.04
- 1.05
- 1.06
- 1.07
- 1.08
- 1.09
- 1.1
- 1.11
- 1.12
- 1.13
- 1.14
- 1.15
- 1.16
- 1.17
- 1.18
- 1.19
- 1.2
- 1.21
- 1.22
- 1.23
- 1.24
- 1.25
- 1.26
- 1.27
- 1.28
- 1.29
- 1.3
- 1.31
- 1.32
- 1.33
- 1.34
- 1.35
- 1.36
- 1.37
- 1.38
- 1.39
- 1.4
- 1.41
- 1.42
- 1.43
- 1.44
- 1.45
- 1.46
- 1.47
- 1.48
- 1.49
- 1.5
- 1.51
- 1.52

Rating vs Complexity Average vs Domain



Rating Average vs. Complexity Average. Color shows details about Domains (group). The view is filtered on Domains (group), which excludes Null.

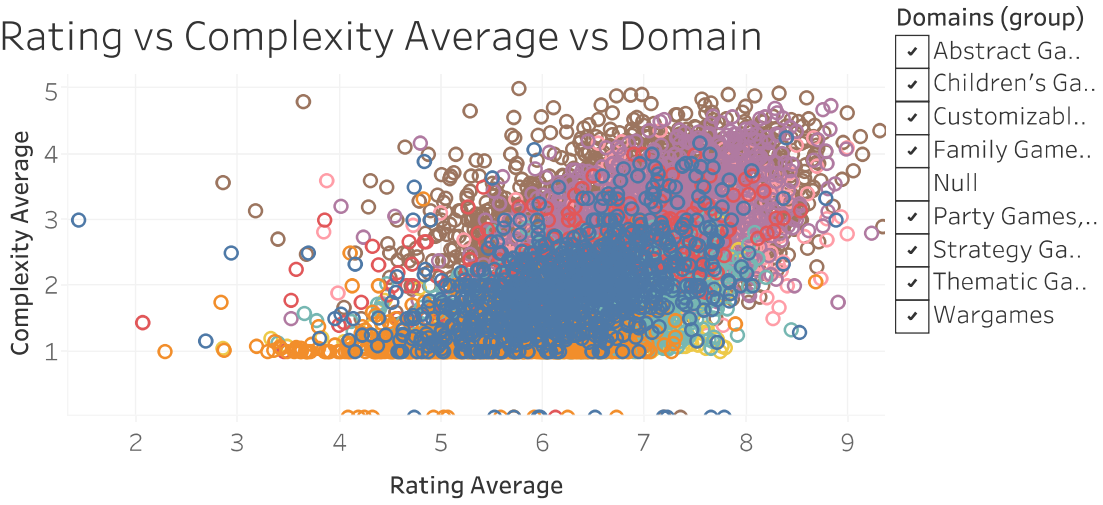


Owned Users vs. Users Rated.

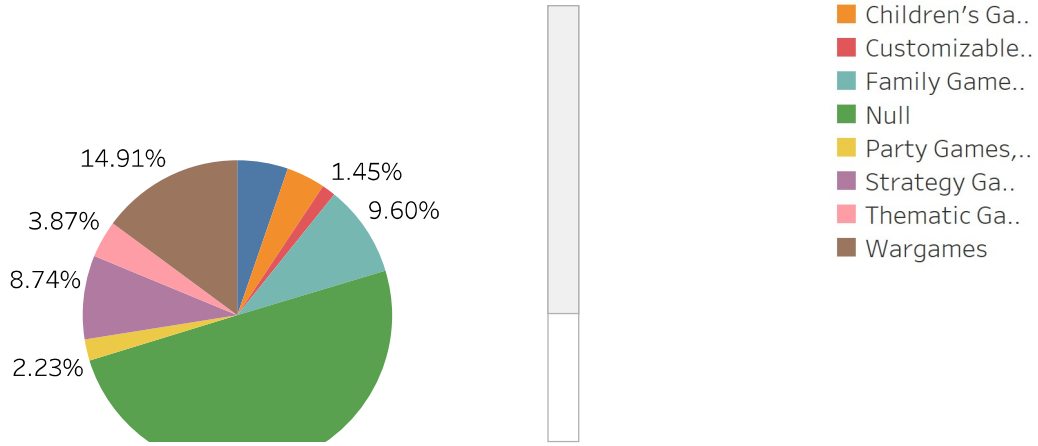
Games Rated

20,320

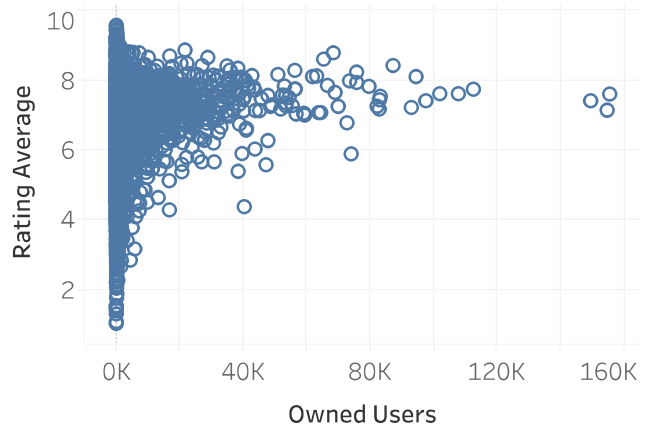
Rating vs Complexity Average vs Domain



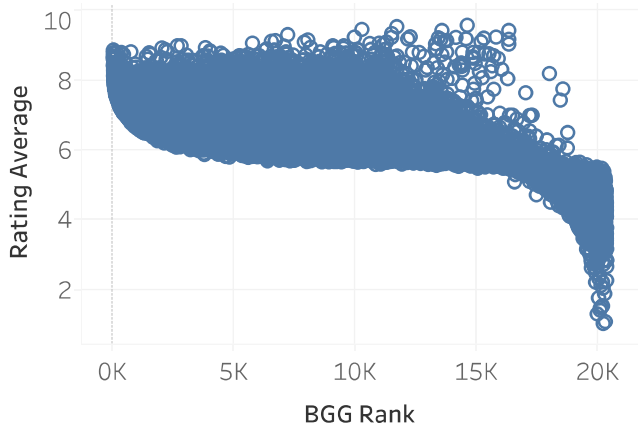
Grouped Domains



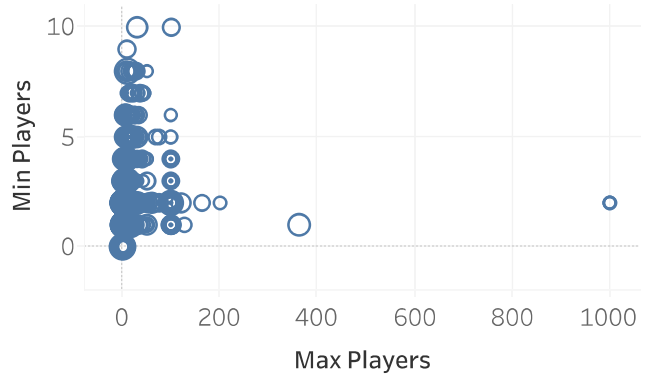
Owned Users vs Rating Average



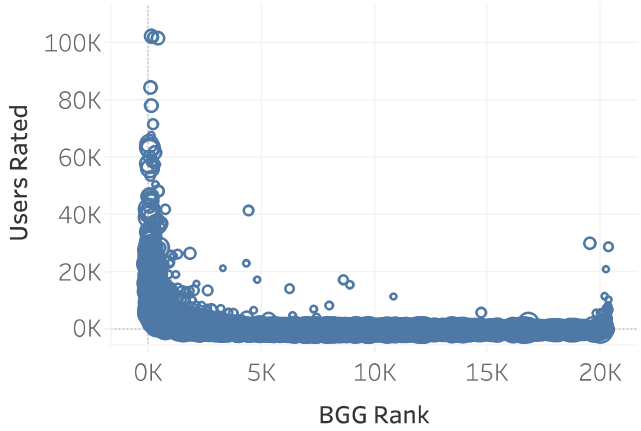
Rank vs Rating Avg



Min-Max players vs Complexity Avg



Rank vs Users Rated

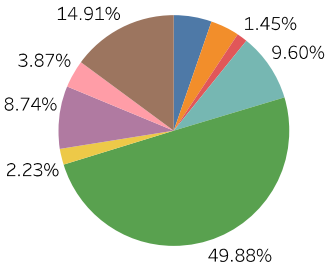


Games Rated

20,320

Count of games.

Grouped Domains



Domains (group)
- Abstract Games, Abstract Games, Children's Games, Abstract Games, Children's Games, Wargames and 7 more
- Children's Games, Children's Games, Family Games, Children's Games, Family Games, Party Games and 2 more
- Customizable Games, Customizable Games, Strategy Games, Customizable Games, Thematic Games and 2 more
- Family Games, Family Games, Party Games, Family Games, Party Games, Thematic Games and 5 more
- Null
- Party Games, Party Games, Strategy Games, Party Games, Thematic Games and 1 more
- Strategy Games, Strategy Games, Thematic Games, Strategy Games, Thematic Games, Wargames and 1 more
- Thematic Games & Thematic Games, Wargames
- Wargames

% of Total Count of games. Color shows details about Domains (group). The marks are labeled by % of Total Count of games.

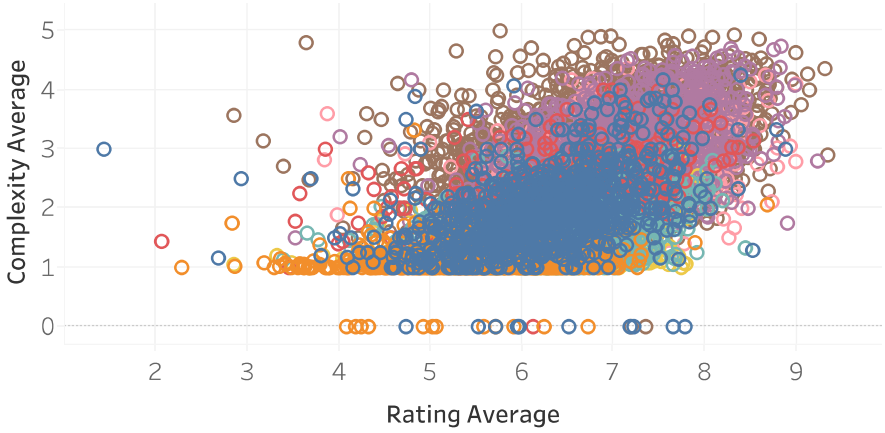
Board Games

Ranking, complexity av..	Exploration	Binning Domains	Scale and run the model	Domains Confusion Ma..	Ratings Logistic Regr..

Games Rated

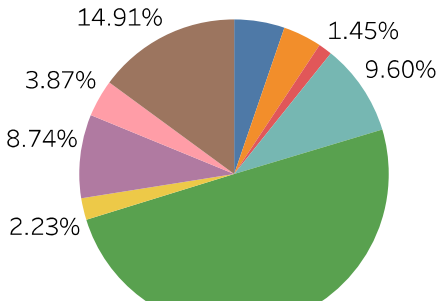
20,320

Rating vs Complexity Average vs Domain



Domains (group)
- ✔ Abstract Ga..
- ✔ Children's Ga..
- ✔ Customizabl..
- ✔ Family Game..
- Null
- ✔ Party Games,..
- ✔ Strategy Ga..
- ✔ Thematic Ga..
- ✔ Wargames

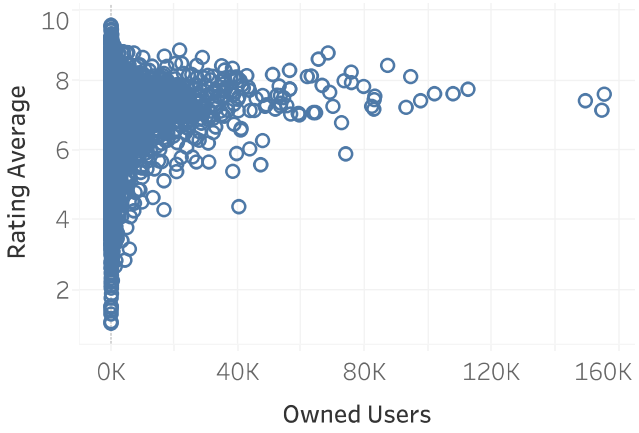
Grouped Domains



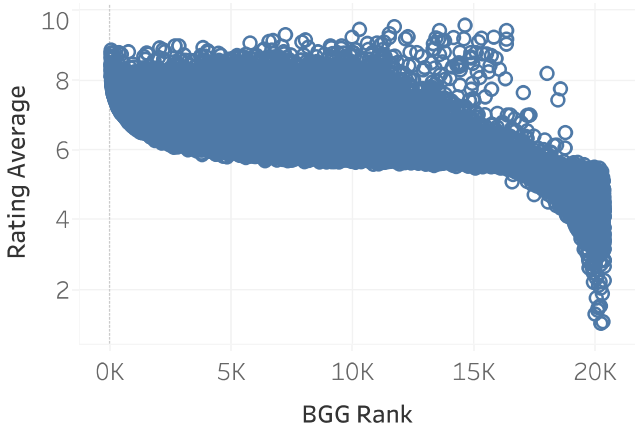
Domains (group)
- Abstract Ga..
- Children's Ga..
- Customizable..
- Family Game..
- Null
- Party Games,..
- Strategy Ga..
- Thematic Ga..
- Wargames

Board Games

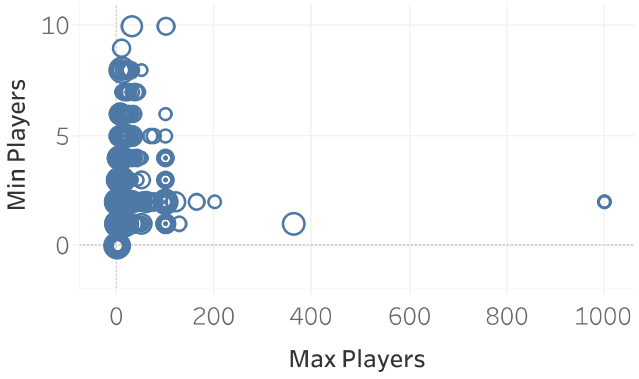
Owned Users vs Rating Average



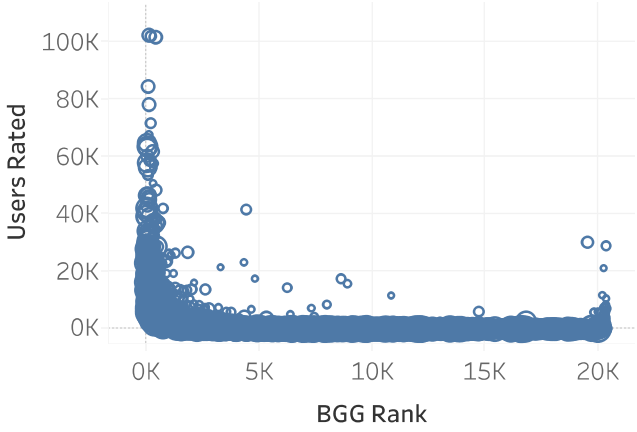
Rank vs Rating Avg



Min-Max players vs Complexity Avg



Rank vs Users Rated



Board Games


```
# 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
```

```
# 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

Board Games


```
# Scale the data
scaler=StandardScaler()
# Fit the scaler with the training data
X_scaler=scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

```
# Instatiate Logistic Regression model
classifier=LogisticRegression(solver='lbfgs',max_iter=500,random_state=1)
```

```
# Train the model
classifier.fit(X_train_scaled, y_train)
```

```
LogisticRegression(max_iter=500, random_state=1)
```

```
# Test the model
y_pred=classifier.predict(X_test_scaled)
```

```
# Check the accuracy score
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
```

```
0.5616653574234093
```

Board Games

	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

Board Games

Ranking, complexity av..	Exploration	Binning Domains	Scale and run the model	Domains Confusion Ma..	Ratings Logistic Regr..

```
# 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
```

	Unnamed: 0	year_published	min_players	max_players	play_time	min_age	users_rated	rating_average	bgg_rank	complexity_average	owned_users	children_games
0	0	2017	1	4	120	14	42.055	8.79	1	3.86	68.323	0
1	1	2015	2	4	60	13	41.643	8.61	2	2.84	65.294	0
2	2	2018	2	4	120	14	19.217	8.66	3	3.91	28.785	0
3	3	2016	1	5	120	12	64.864	8.43	4	3.24	87.099	0
4	4	2017	3	6	480	14	13.468	8.70	5	4.22	16.831	0

```
# label variables
y= games_binned_df["rating_bins"]
X= games_binned_df.drop(columns="rating_bins")
```

```
# Split
X_train, X_test, y_train, y_test = train_test_split(X,
    y, random_state=1, stratify=y)
```

```
# Model
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(solver='lbfgs',
    max_iter=200,
    random_state=1)
```

```
# Train
classifier.fit(X_train, y_train)
```

```
# get accuracy score
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.7836614173228347
```

```
from sklearn.metrics import confusion_matrix, classification_report
matrix = confusion_matrix(y_test, y_pred)
print(matrix)
```

```
[[1934  547]
 [ 552 2047]]
```

```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.78      0.78      0.78      2481
           1       0.79      0.79      0.79      2599

    accuracy                           0.78      5080
   macro avg       0.78      0.78      0.78      5080
weighted avg       0.78      0.78      0.78      5080
```

```
# 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
```

```
# 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

```
# Scale the data
scaler=StandardScaler()
# Fit the scaler with the training data
X_scaler=scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

```
# Instatiate Logistic Regression model
classifier=LogisticRegression(solver='lbfgs',max_iter=500,random_state=1)
```

```
# Train the model
classifier.fit(X_train_scaled, y_train)
```

```
LogisticRegression(max_iter=500, random_state=1)
```

```
# Test the model
y_pred=classifier.predict(X_test_scaled)
```

```
# Check the accuracy score
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
```

```
0.5616653574234093
```

	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

```
# 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
```

	Unnamed: 0	year_published	min_players	max_players	play_time	min_age	users_rated	rating_average	bgg_rank	complexity_average	owned_users	children_games
0	0	2017	1	4	120	14	42.055	8.79	1	3.86	68.323	0
1	1	2015	2	4	60	13	41.643	8.61	2	2.84	65.294	0
2	2	2018	2	4	120	14	19.217	8.66	3	3.91	28.785	0
3	3	2016	1	5	120	12	64.864	8.43	4	3.24	87.099	0
4	4	2017	3	6	480	14	13.468	8.70	5	4.22	16.831	0

```
# label variables
y= games_binned_df["rating_bins"]
X= games_binned_df.drop(columns="rating_bins")
```

```
# Split
X_train, X_test, y_train, y_test = train_test_split(X,
   y, random_state=1, stratify=y)
```

```
# Model
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(solver='lbfgs',
   max_iter=200,
   random_state=1)
```

```
# Train
classifier.fit(X_train, y_train)
```

```
# get accuracy score
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.7836614173228347
```

```
from sklearn.metrics import confusion_matrix, classification_report
matrix = confusion_matrix(y_test, y_pred)
print(matrix)
```

```
[[1934  547]
 [ 552 2047]]
```

```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.78      0.78      0.78      2481
           1       0.79      0.79      0.79      2599

    accuracy                           0.78      5080
   macro avg       0.78      0.78      0.78      5080
weighted avg       0.78      0.78      0.78      5080
```