Feature Engineering

Feature selection methods are intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable. We have performed the feature selection techniques highlighted in bold below.

We can summarize feature selection as follows.

- Feature Selection: Select a subset of input features from the dataset.
    - Unsupervised: Do not use the target variable (e.g. remove redundant variables).
        - Correlation – **Pearson Correlation Coefficient, Variance Threshold**
    - Supervised: Use the target variable (e.g. remove irrelevant variables).
        - Wrapper: Search for well-performing subsets of features.
            - RFE
        - Filter: Select subsets of features based on their relationship with the target.
            - Statistical Methods – **Univariate Selection**
            - Feature Importance Methods
        - Intrinsic: Algorithms that perform automatic feature selection during training.
            - Decision Trees
- Dimensionality Reduction: Project input data into a lower-dimensional feature space.

Filter Selection methods

- Filter methods: The features are ranked by the score and either selected to be kept or removed from the dataset.
    - Basic filter methods, correlation coefficient scores, Chi-squared test, mutual information
- Wrapper methods: The selection of a set of features as a search problem, where different combinations are prepared, evaluated and compared to other combinations.
    - Forward selection, backward selection, exhaustive feature selection
- Embedded methods–Combine the advantages of both methods–Include the feature selection process in the machine learning model training
    - Regularization methods such as LASSO, elastic net and ridge regression, decision tree algorithms (giniindex, information gain)
- Wrapper/Embedded methods
    - Recursive feature elimination–Recursive feature elimination with CV

Filter Methods

Select features independently of the machine learning algorithm model

- Basic filter methods
    - Remove constant features
    - Remove quasi-constant features
    - Remove duplicated features

- Correlation methods
    - Features are selected on the basis of their scores between their correlations
    - correlation coefficient scores
- Statistical ranking filter methods
    - Features are statistically ranked by the score and either selected to be kept or removed from the dataset
    - Chi-squared test, mutual information

Variance Threshold

Variance Threshold removes features with a variance less than the specified threshold. Consider a feature that takes the same value for all the observations (rows) in the dataset. It would not add any informative power to a model. Using this feature also adds an unnecessary computation burden. Thus, we should just eliminate it from the dataset. Similarly, features with a very small variance can also be omitted.

sklearn.feature_selection.VarianceThreshold is a feature selector that removes all low-variance features. This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

#Variance Threshold

#Does not require target feature hence dropping Board Game Rank Feature selector that removes all low-variance features. This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

In [60]:
```
df1 = pd.read_csv('modified-games_detailed_info.csv')
df1 = df1.drop(['id', 'primary', 'description', 'yearpublished','boardgamecategory','boardgamemechanic','boardga
df1 = df1.drop(['Board Game Rank'], axis=1)
df1.head(4)
```

Out[60]:

	minplayers	maxplayers	playingtime	minplaytime	maxplaytime	minage	usersrated	average
0	2	4	45	45	45	8	96241	7.61567
1	2	5	45	30	45	7	96228	7.41884
2	3	4	120	60	120	10	96213	7.16265
3	2	7	30	30	30	10	79873	7.76049

In [17]:
```
from sklearn.feature_selection import VarianceThreshold
constant_filter=VarianceThreshold(threshold=0)
constant_filter.fit(df1)
```

Out[17]:
```
VarianceThreshold(threshold=0)
```

In [18]:
```
len(df1.columns[constant_filter.get_support()])
```

Out[18]:
```
8
```

In [19]:
```
constant_filter.get_support()
```

Out[19]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True])
```

In [21]:
```
df1.columns[constant_filter.get_support()]
```

Out[21]:
```
Index(['minplayers', 'maxplayers', 'playingtime', 'minplaytime', 'maxplaytime',
       'minage', 'usersrated', 'average'],
      dtype='object')
```

In [22]:
```
constant_columns = [column for column in df1.columns
                    if column not in df1.columns[constant_filter.get_support()]]

print(len(constant_columns))
```

```
0
```

In [23]:
```
for feature in constant_columns:
    print(feature)
```

In [24]:
```
df1
```

Out[24]:

	minplayers	maxplayers	playingtime	minplaytime	maxplaytime	minage	usersrated	average
0	2	4	45	45	45	8	96241	7.61567
1	2	5	45	30	45	7	96228	7.41884
2	3	4	120	60	120	10	96213	7.16265

```
In [25]: 1 df1_T = df1.T

In [26]: 1 df1_T
```

Out[26]:

	0	1	2	3	4	5	6	7	8	9	...	19220
minplayers	2.00000	2.00000	3.00000	2.00000	2.00000	2.0000	2.00000	1.00000	3.00000	2.00000	...	2.00000
maxplayers	4.00000	5.00000	4.00000	7.00000	4.00000	5.0000	8.00000	5.00000	5.00000	5.00000	...	2.00000
playingtime	45.00000	45.00000	120.00000	30.00000	30.00000	60.0000	15.00000	150.00000	150.00000	80.00000	...	360.00000
minplaytime	45.00000	30.00000	60.00000	30.00000	30.00000	30.0000	15.00000	30.00000	90.00000	40.00000	...	360.00000
maxplaytime	45.00000	45.00000	120.00000	30.00000	30.00000	60.0000	15.00000	150.00000	150.00000	80.00000	...	360.00000
minage	8.00000	7.00000	10.00000	10.00000	13.00000	8.0000	14.00000	12.00000	12.00000	8.00000	...	14.00000
usersrated	96241.00000	96228.00000	96213.00000	79873.00000	74959.00000	67987.0000	62582.00000	61565.00000	61164.00000	59221.00000	...	30.00000
average	7.61567	7.41884	7.16265	7.76049	7.62671	7.4299	7.63806	7.95898	8.00613	7.27135	...	8.38333

8 rows × 19230 columns

```
In [27]: 1 df1_T.duplicated()
```

Out[27]:
```
minplayers     False
maxplayers     False
playingtime    False
minplaytime    False
maxplaytime     True
minage         False
usersrated     False
average        False
dtype: bool
```

```
In [28]: 1 duplicated_columns = df1_T[df1_T.duplicated()].index.values

In [29]: 1 duplicated_columns
```

Out[29]:
```
array(['maxplaytime'], dtype=object)
```

```
In [30]: 1 df1.drop(labels = duplicated_columns,axis = 1, inplace = True)

In [31]: 1 df1
```

Out[31]:

	minplayers	maxplayers	playingtime	minplaytime	minage	usersrated	average
0	2	4	45	45	8	96241	7.61567
1	2	5	45	30	7	96228	7.41884
2	3	4	120	60	10	96213	7.16265
3	2	7	30	30	10	79873	7.76049
4	2	4	30	30	13	74959	7.62671
5	2	5	60	30	8	67987	7.42990
6	2	8	15	15	14	62582	7.63806
7	1	5	150	30	12	61565	7.95898
8	3	5	150	90	12	61164	8.00613
9	2	5	80	40	8	59221	7.27135
10	1	5	120	120	12	57395	8.43086

- After processing, we can conclude that column ‘maxplaytime’ had the least variance and hence was dropped from the dataset.

Pearson Correlation Coefficient

A Pearson correlation is a number between -1 and 1 that indicates the extent to which two variables are linearly related. The Pearson correlation is also known as the “product moment correlation coefficient” (PMCC) or simply “correlation”. Pearson correlations are suitable only for metric variables.
The correlation coefficient has values between -1 to 1

- A value closer to 0 implies weaker correlation (exact 0 implying no correlation)
- A value closer to 1 implies stronger positive correlation
- A value closer to -1 implies stronger negative correlation

In Pearson correlation coefficient:

- Both variables are normally distributed (Gaussian distributions)
- A straight-line relationship between the two variables
- Data is distributed around the regression line
- Pearson correlation coefficient and linear regression are highly correlated

In [ ]:
```
#Pearson corelation
```

In [32]:
```
df1 = pd.read_csv('modified-games_detailed_info.csv')
df1 = df1.drop(['id', 'primary', 'description', 'yearpublished','boardgamecategory','boardgamemechanic','boardga
```

In [34]:
```
df1.corr()
```

Out[34]:

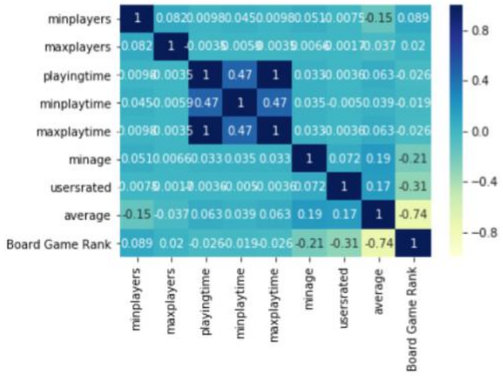
	minplayers	maxplayers	playingtime	minplaytime	maxplaytime	minage	usersrated	average	Board Game Rank
minplayers	1.000000	0.082396	0.009832	0.045492	0.009832	0.051410	-0.007511	-0.152032	0.089065
maxplayers	0.082396	1.000000	-0.003462	-0.005878	-0.003462	0.006551	-0.001672	-0.036809	0.020347
playingtime	0.009832	-0.003462	1.000000	0.470732	1.000000	0.032717	-0.003579	0.063192	-0.025746
minplaytime	0.045492	-0.005878	0.470732	1.000000	0.470732	0.034672	-0.005010	0.038794	-0.018942
maxplaytime	0.009832	-0.003462	1.000000	0.470732	1.000000	0.032717	-0.003579	0.063192	-0.025746
minage	0.051410	0.006551	0.032717	0.034672	0.032717	1.000000	0.071956	0.189741	-0.212169
usersrated	-0.007511	-0.001672	-0.003579	-0.005010	-0.003579	0.071956	1.000000	0.173336	-0.312941
average	-0.152032	-0.036809	0.063192	0.038794	0.063192	0.189741	0.173336	1.000000	-0.741880
Board Game Rank	0.089065	0.020347	-0.025746	-0.018942	-0.025746	-0.212169	-0.312941	-0.741880	1.000000

In [39]:
```
import seaborn as sns
```

In [39]:
```
1 import seaborn as sns
```

In [43]:
```
1 sns.heatmap(df1.corr(),vmin = -1, vmax = 1, cmap = "YlGnBu", annot = True)
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb5c11c5358>



In [45]:
```
1 corr_matrix = df1.corr()
```

In [46]:
```
1 corr_matrix
```

Out[46]:

	minplayers	maxplayers	playingtime	minplaytime	maxplaytime	minage	usersrated	average	Board Game Rank
minplayers	1.000000	0.082396	0.009832	0.045492	0.009832	0.051410	-0.007511	-0.152032	0.089065
maxplayers	0.082396	1.000000	-0.003462	-0.005878	-0.003462	0.006551	-0.001672	-0.036809	0.020347
playingtime	0.009832	-0.003462	1.000000	0.470732	1.000000	0.032717	-0.003579	0.063192	-0.025746
minplaytime	0.045492	-0.005878	0.470732	1.000000	0.470732	0.034672	-0.005010	0.038794	-0.018942
maxplaytime	0.009832	-0.003462	1.000000	0.470732	1.000000	0.032717	-0.003579	0.063192	-0.025746
minage	0.051410	0.006551	0.032717	0.034672	0.032717	1.000000	0.071956	0.189741	-0.212169
usersrated	-0.007511	-0.001672	-0.003579	-0.005010	-0.003579	0.071956	1.000000	0.173336	-0.312941
average	-0.152032	-0.036809	0.063192	0.038794	0.063192	0.189741	0.173336	1.000000	-0.741880
Board Game Rank	0.089065	0.020347	-0.025746	-0.018942	-0.025746	-0.212169	-0.312941	-0.741880	1.000000

In [51]:
```
1  #Pearson correlation coefficient
2  #creating set to hold the corelated features
3
4  corr_features = set()
5  corr_threshold = 0.5
6
7  for i in range(len(corr_matrix.columns)):
8      for j in range(i):
9          if abs(corr_matrix.iloc[i, j]) > corr_threshold: # we are interested in absolute coeff value
10             colname = corr_matrix.columns[i]  # getting the name of column
11             corr_features.add(colname)
```

In [52]:
```
1 corr_features
```

Out[52]: {'Board Game Rank', 'maxplaytime'}

From the above results , we can observe that the feature ‘Board Game Rank’ and ‘maxplaytime’ are somewhat corelated.

Univariate Selection

Univariate feature selection works by selecting the best features based on univariate statistical tests. We compare each feature to the target variable, to see whether there is any statistically significant relationship between them. It is also called analysis of variance (ANOVA). When we analyze the relationship between one feature and the target variable, we ignore the other features. That is why it is called 'univariate'.

Each feature has its test score. Finally, all the test scores are compared, and the features with top scores will be selected. Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features.

The example below uses the chi-squared (chi^2) statistical test for non-negative features to select 10 of the best features from the Mobile Price Range Prediction Dataset. It is very similar to Spearman's rank correlation coefficient.

In [13]:
```
#Deleting all text columns and keeping only numerical data
df1 = df1.drop(['id', 'primary', 'description', 'yearpublished','boardgamecategory','boardgamemechanic','boardga
df1
```

Out[13]:

	minplayers	maxplayers	playingtime	minplaytime	maxplaytime	minage	usersrated	average	Board Game Rank
0	2	4	45	45	45	8	96241	7.61567	91
1	2	5	45	30	45	7	96228	7.41884	173
2	3	4	120	60	120	10	96213	7.16265	381
3	2	7	30	30	30	10	79873	7.76049	51
4	2	4	30	30	30	13	74959	7.62671	89
5	2	5	60	30	60	8	67987	7.42990	166
6	2	8	15	15	15	14	62582	7.63806	80
7	1	5	150	30	150	12	61565	7.95898	29
8	3	5	150	90	150	12	61164	8.00613	26
9	2	5	80	40	80	8	59221	7.27135	239
10	1	5	120	120	120	12	57395	8.43086	4

In [11]:
```
#Univariate Selection
df1 = pd.read_csv('modified-games_detailed_info.csv')
df1.head(4)
print(df1.shape)
```

```
(19230, 20)
```

In [12]:
```
df1.head(4)
```

Out[12]:

	id	primary	description	yearpublished	minplayers	maxplayers	playingtime	minplaytime	maxplaytime	minage	boardgamecategory	boardgamen
0	30549	Pandemic	In Pandemic, several virulent diseases have br...	2008	2	4	45	45	45	8	['Medical']	['Actio 'Cooperati 'H
1	822	Carcassonne	Carcassonne is a tile-placement game in which ...	2000	2	5	45	30	45	7	['City Building', 'Medieval', 'Territory Build...	['Area Influen Ad
2	13	Catan	In Catan (formerly The Settlers of Catan), pla...	1995	3	4	120	60	120	10	['Economic', 'Negotiation']	['Dic 'Hexa 'Incor
3	68448	7 Wonders	You are the leader of one of the 7 great citie...	2010	2	7	30	30	30	10	['Ancient', 'Card Game', 'City Building', 'Civ...	['Card 'Draftin Mana

In [57]:
```
X = df1.iloc[:,0:7]  #independent columns
y = df1.iloc[:,-1]    #target column i.e Board Game Rank
```

In [58]:
```
#apply SelectKBest class to extract top k best features
bestfeatures = SelectKBest(score_func=chi2, k=7)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
```

In [59]:
```
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(7,'Score'))  #print k best features
```

```
        Specs         Score
6   usersrated  2.669023e+08
2  playingtime  2.094903e+08
4  maxplaytime  2.094903e+08
3  minplaytime  6.473351e+07
1   maxplayers  8.087590e+05
5       minage  2.640644e+04
0   minplayers  4.456536e+03
```

- Since we have 7 features we will try to select just 7 best features.
- From the above score we can see that ‘usersrated’ and ‘playingtime’ are directly corelated to Board Game Rank and so on in that order. So more the number of users rated and more the playing time, higher will be the Game rank.
- A good board game rank will specify a smaller value Ex. Game at rank 1 will be better than game at rank 78. So less the playing time of game, less or better will be the board game rank.

- We can make some inference from these results if we wish to design a game with good Board Game rank
    - Keep the playing time and maximum playing time of game short since it directly affects the board game rank.
    - Minimum players and minimum age would not have much impact on the board game rank.