FOCUS TESTING

15IT322E- PYTHON PROGRAMMING PROJECT REPORT

Submitted by

S.MAHADEVAN (RA1511003010422)

For the assessment of Semester V Minor Project

SRM UNIVERSITY

KATTANKULATHUR

NOV 2017

DECLARATION

I ……S.MAHADEVAN, RA1511003010422,……..studying in III year B.Tech Computer Science program at SRM University, Kattankulathur, Chennai, hereby declare that this project is an original work of mine and I have not verbatim copied / duplicated any material from sources like internet or from print media, excepting some vital company information / statistics and data that is provided by the company itself.

Signature of the Student

Date:

Place:

ACKNOWLEDGEMENT

This Python Project was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to do it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this training period.

Bearing this in mind, I am using this opportunity to express my deepest gratitude and special thanks to my Python subject professor, ***Ms. T Krithiga Devi***, who in spite of being extraordinarily busy with her duties, took time out to hear, guide and keep me on the correct path while making the project.

I would also like to thank all my friends and colleagues who supported me all the way through the project by offering assistance and providing constructive criticism. Without them, this project would not have been possible.

I perceive this opportunity as a big milestone in my career development. I will strive to use my gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain my desired career objectives. I hope to continue cooperation with all of you in the future,

Sincerely,

S.Mahadevan

ABSTRACT

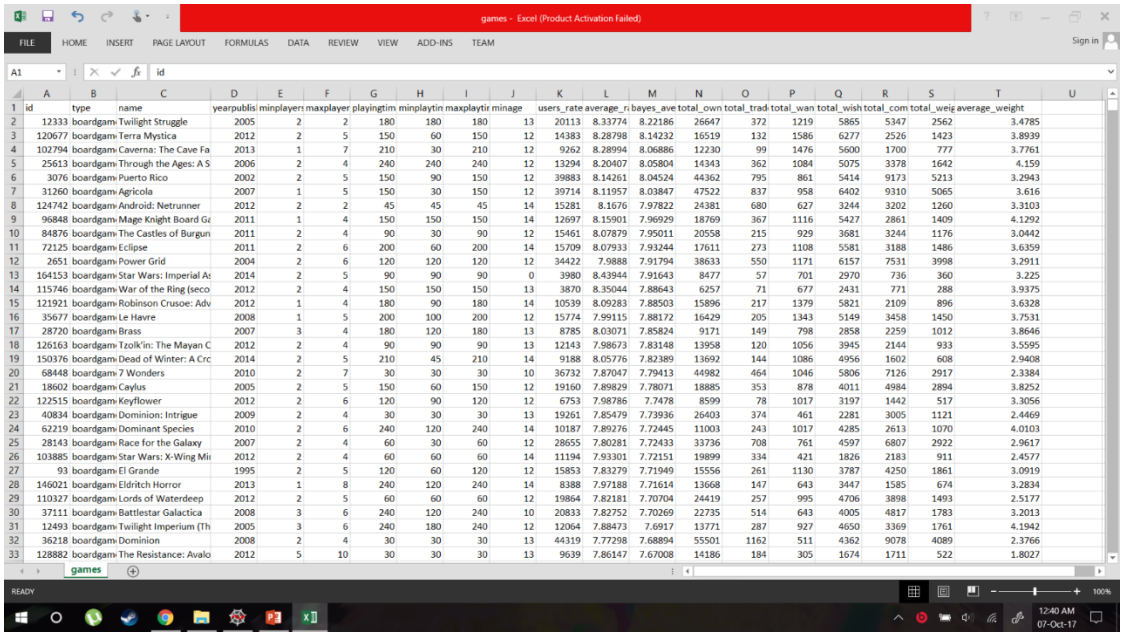
This project is intended to be a proof of concept of a **focus testing application**, using machine learning. Given a board game database, we are going to **predict the average user rating** a new board game might get, based on a number of different fields such as year published, maximum and minimum players required to play the game, average weight etc.

The application accepts values of different fields from the user, in a GUI interface, and makes a prediction of the rating it would get.

This application can have various applications in various product development fields like **mobile, software development, automobile** etc. With enough refinement, this application can be extrapolated to different product development fields such as mobile computing, software development and automobile industry. It could save a company billions of dollars it spends on focus testing with humans. By making this entire process automated, we also **eliminate any bias associated with human testing**. It will also help the company make more informed decisions about their products and how it would fare in the market.

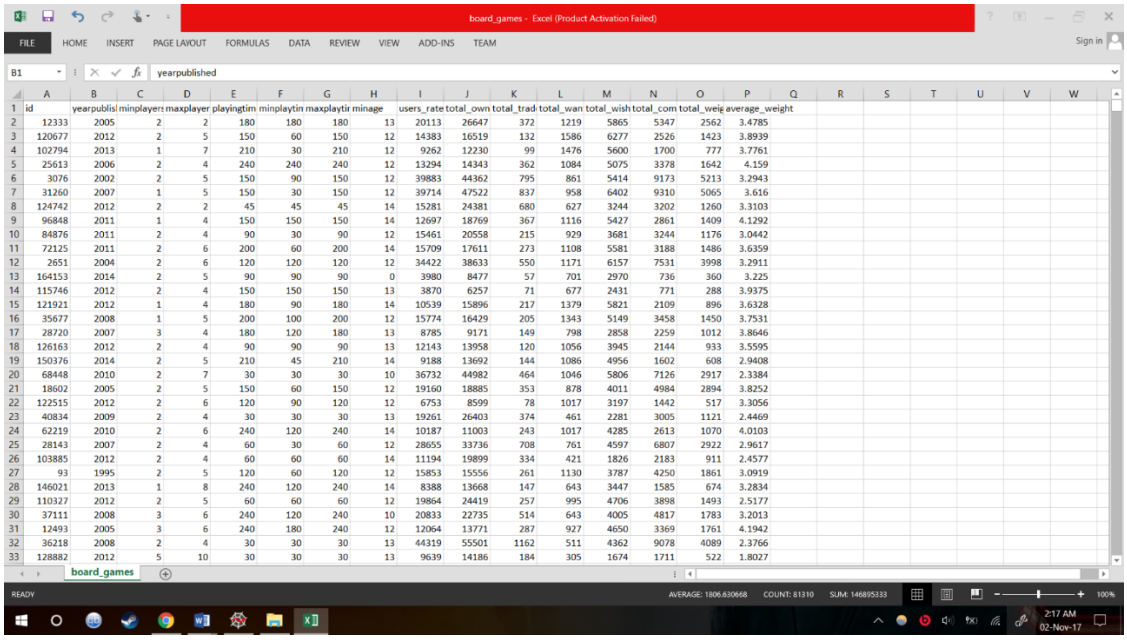
TABLE OF CONTENTS

LIST OF TABLES



id	type	name	yearpublis	minplayers	maxplayer	playingtim	minplaytin	maxplaytir	minage	users_rate	average_ri	bayes_ave	total_own	total_trad	total_wan	total_wish	total_com	total_weig	average_weight
12333	boardgam	Twilight Struggle	2005	2	2	180	180	180	13	20113	8.33774	8.22186	26647	372	1219	5865	5347	2562	3.4785
120677	boardgam	Terra Mystica	2012	2	5	150	60	150	12	14383	8.28798	8.14232	16519	132	1586	6277	2526	1423	3.8939
102794	boardgam	Caverna: The Cave Fa	2013	1	7	210	30	210	12	9262	8.28994	8.06886	12230	99	1476	5600	1700	777	3.7761
25613	boardgam	Through the Ages: A S	2006	2	4	240	240	240	12	13294	8.20407	8.05804	14343	362	1084	5075	3378	1642	4.159
3076	boardgam	Puerto Rico	2002	2	5	150	90	150	12	39883	8.14261	8.04524	44362	795	861	5414	9173	5213	3.2943
31260	boardgam	Agricola	2007	1	5	150	30	150	12	39714	8.11957	8.03847	47522	837	958	6402	9310	5065	3.616
124742	boardgam	Android: Netrunner	2012	2	2	45	45	45	14	15281	8.1676	7.97822	24381	680	627	3244	3202	1260	3.3103
96848	boardgam	Mage Knight Board Ga	2011	1	4	150	150	150	14	12697	8.15901	7.96929	18769	367	1116	5427	2861	1409	4.1292
84876	boardgam	The Castles of Burgun	2011	2	4	90	30	90	12	15461	8.07879	7.95011	20558	215	929	3681	3244	1176	3.0442
72125	boardgam	Eclipse	2011	2	6	200	60	200	14	15709	8.07933	7.93244	17611	273	1108	5581	3188	1486	3.6359
2651	boardgam	Power Grid	2004	2	6	120	120	120	12	34422	7.9888	7.91794	38633	550	1171	6157	7531	3998	3.2911
164153	boardgam	Star Wars: Imperial As	2014	2	5	90	90	90	0	3980	8.43944	7.91643	8477	57	701	2970	736	360	3.225
115746	boardgam	War of the Ring (seco	2012	2	4	150	150	150	13	3870	8.35044	7.88643	6257	71	677	2431	771	288	3.9375
121921	boardgam	Robinson Crusoe: Adv	2012	1	4	180	90	180	14	10539	8.09283	7.88503	15896	217	1379	5821	2109	896	3.6328
35677	boardgam	Le Havre	2008	1	5	200	100	200	12	15774	7.99115	7.88172	16429	205	1343	5149	3458	1450	3.7531
28720	boardgam	Brass	2007	3	4	180	120	180	13	8785	8.03071	7.85824	9171	149	798	2858	2259	1012	3.8646
126163	boardgam	Tzolk'in: The Mayan C	2012	2	4	90	90	90	13	12143	7.98673	7.83148	13958	120	1056	3945	2144	933	3.5595
150376	boardgam	Dead of Winter: A Cro	2014	2	5	210	45	210	14	9188	8.05776	7.82389	13692	144	1086	4956	1602	608	2.9408
68448	boardgam	7 Wonders	2010	2	7	30	30	30	10	36732	7.87047	7.79413	44982	464	1046	5806	7126	2917	2.3384
18602	boardgam	Caylus	2005	2	5	150	60	150	12	19160	7.89829	7.78071	18885	353	878	4011	4984	2894	3.8252
122515	boardgam	Keyflower	2012	2	6	120	90	120	12	6753	7.98786	7.7478	8599	78	1017	3197	1442	517	3.3056
40834	boardgam	Dominion: Intrigue	2009	2	4	30	30	30	13	19261	7.85479	7.73936	26403	374	461	2281	3005	1121	2.4469
62219	boardgam	Dominant Species	2010	2	6	240	120	240	14	10187	7.89276	7.72445	11003	243	1017	4285	2613	1070	4.0103
28143	boardgam	Race for the Galaxy	2007	2	4	60	30	60	12	28655	7.80281	7.72433	33736	708	761	4597	6807	2922	2.9617
103885	boardgam	Star Wars: X-Wing Mi	2012	2	4	60	60	60	14	11194	7.93301	7.72151	19899	334	421	1826	2183	911	2.4577
93	boardgam	El Grande	1995	2	5	120	60	120	12	15853	7.83279	7.71949	15556	261	1130	3787	4250	1861	3.0919
146021	boardgam	Eldritch Horror	2013	1	8	240	120	240	14	8388	7.97188	7.71614	13668	147	643	3447	1585	674	3.2834
110327	boardgam	Lords of Waterdeep	2012	2	5	60	60	60	12	19864	7.82181	7.70704	24419	257	995	4706	3898	1493	2.5177
37111	boardgam	Battlestar Galactica	2008	3	6	240	120	240	10	20833	7.82752	7.70269	22735	514	643	4005	4817	1783	3.2013
12493	boardgam	Twilight Imperium (Th	2005	3	6	240	180	240	12	12064	7.88473	7.6917	13771	287	927	4650	3369	1761	4.1942
36218	boardgam	Dominion	2008	2	4	30	30	30	13	44319	7.77298	7.68894	55501	1162	511	4362	9078	4089	2.3766
128882	boardgam	The Resistance: Avalo	2012	5	10	30	30	30	13	9639	7.86147	7.67008	14186	184	305	1674	1711	522	1.8027

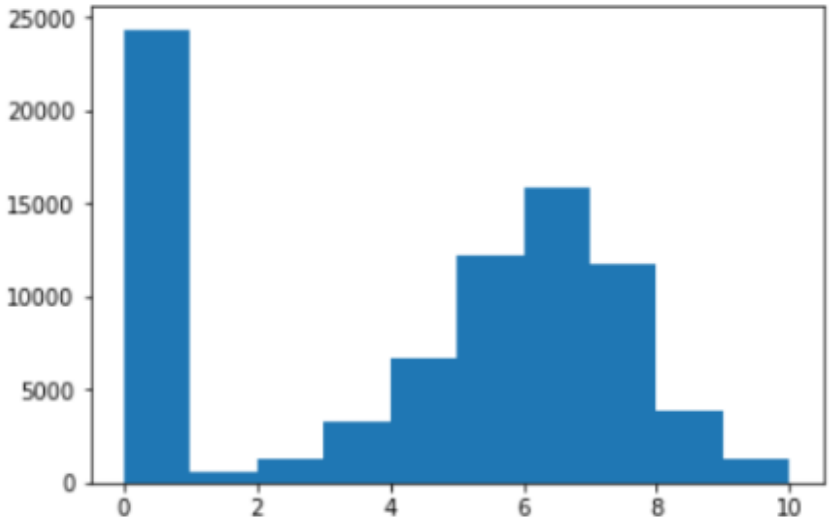
A snapshot of the board game database to be used for the project



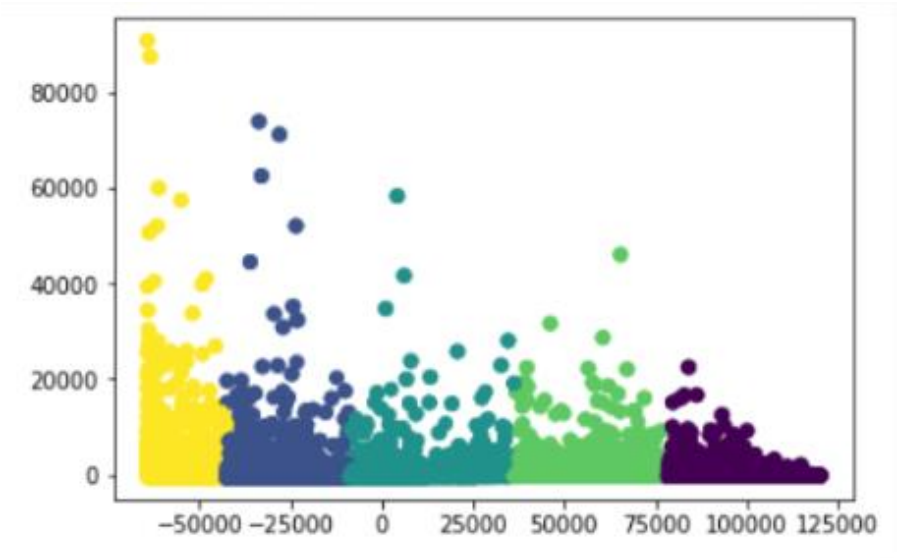
id	yearpublis	minplayers	maxplayer	playingtim	minplaytin	maxplaytir	minage	users_rate	total_own	total_trad	total_wan	total_wish	total_com	total_weig	average_weight
12333	2005	2	2	180	180	180	13	20113	26647	372	1219	5865	5347	2562	3.4785
120677	2012	2	5	150	60	150	12	14383	16519	132	1586	6277	2526	1423	3.8939
102794	2013	1	7	210	30	210	12	9262	12230	99	1476	5600	1700	777	3.7761
25613	2006	2	4	240	240	240	12	13294	14343	362	1084	5075	3378	1642	4.159
3076	2002	2	5	150	90	150	12	39883	44362	795	861	5414	9173	5213	3.2943
31260	2007	1	5	150	30	150	12	39714	47522	837	958	6402	9310	5065	3.616
124742	2012	2	2	45	45	45	14	15281	24381	680	627	3244	3202	1260	3.3103
96848	2011	1	4	150	150	150	14	12697	18769	367	1116	5427	2861	1409	4.1292
84876	2011	2	4	90	30	90	12	15461	20558	215	929	3681	3244	1176	3.0442
72125	2011	2	6	200	60	200	14	15709	17611	273	1108	5581	3188	1486	3.6359
2651	2004	2	6	120	120	120	12	34422	38633	550	1171	6157	7531	3998	3.2911
164153	2014	2	5	90	90	90	0	3980	8477	57	701	2970	736	360	3.225
115746	2012	2	4	150	150	150	13	3870	6257	71	677	2431	771	288	3.9375
121921	2012	1	4	180	90	180	14	10539	15896	217	1379	5821	2109	896	3.6328
35677	2008	1	5	200	100	200	12	15774	16429	205	1343	5149	3458	1450	3.7531
28720	2007	3	4	180	120	180	13	8785	9171	149	798	2858	2259	1012	3.8646
126163	2012	2	4	90	90	90	13	12143	13958	120	1056	3945	2144	933	3.5595
150376	2014	2	5	210	45	210	14	9188	13692	144	1086	4956	1602	608	2.9408
68448	2010	2	7	30	30	30	10	36732	44982	464	1046	5806	7126	2917	2.3384
18602	2005	2	5	150	60	150	12	19160	18885	353	878	4011	4984	2894	3.8252
122515	2012	2	6	120	90	120	12	6753	8599	78	1017	3197	1442	517	3.3056
40834	2009	2	4	30	30	30	13	19261	26403	374	461	2281	3005	1121	2.4469
62219	2010	2	6	240	120	240	14	10187	11003	243	1017	4285	2613	1070	4.0103
28143	2007	2	4	60	30	60	12	28655	33736	708	761	4597	6807	2922	2.9617
103885	2012	2	4	60	60	60	14	11194	19899	334	421	1826	2183	911	2.4577
93	1995	2	5	120	60	120	12	15853	15556	261	1130	3787	4250	1861	3.0919
146021	2013	1	8	240	120	240	14	8388	13668	147	643	3447	1585	674	3.2834
110327	2012	2	5	60	60	60	12	19864	24419	257	995	4706	3898	1493	2.5177
37111	2008	3	6	240	120	240	10	20833	22735	514	643	4005	4817	1783	3.2013
12493	2005	3	6	240	180	240	12	12064	13771	287	927	4650	3369	1761	4.1942
36218	2008	2	4	30	30	30	13	44319	55501	1162	511	4362	9078	4089	2.3766
128882	2012	5	10	30	30	30	13	9639	14186	184	305	1674	1711	522	1.8027

The final table to be used for testing, with all redundant columns removed

LIST OF FIGURES



A histogram showing the games with their average rating, with number of games on the y axis, and the rating it received on the x axis.



A scatter plot of the columns in the database, reduced to 2 dimensions, using PCA

```
Out[15]: id                     0.304201
         yearpublished          0.108461
         minplayers            -0.032701
         maxplayers            -0.008335
         playingtime            0.048994
         minplaytime            0.043985
         maxplaytime            0.048994
         minage                 0.210049
         users_rated            0.112564
         average_rating         1.000000
         bayes_average_rating   0.231563
         total_owners           0.137478
         total_traders          0.119452
         total_wanters          0.196566
         total_wishers          0.171375
         total_comments         0.123714
         total_weights          0.109691
         average_weight         0.351081
         Name: average_rating, dtype: float64
```

Correlation table between average rating and the other fields

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

Machine Learning	– ML
Principal Component Analysis	– PCA
Scikit- Learn	– SKL
Comma separated values	– CSV
Dataframe	– DF
Support Vector Machine	– SVM
K-Means Clustering	– KMC
Mean Square Error	– MSE

INTRODUCTION

GENERAL

ML is a field that uses algorithms to **learn from data and make predictions**. Practically, this means that we can feed data into an algorithm, and use it to make predictions about what might happen in the future. This has a vast range of applications, from self-driving cars to stock price prediction. Not only is ML interesting, it's also starting to be widely used.

This project is about finding patterns in a board game database. The database contains **81312 rows and 20 column attributes**. The dataset is from BoardGameGeek, and contains data on 80000 board games.

By the end of this project, we would have been able to predict the average rating of a board game and would have gained an understanding on creating focus testing applications for other industries.

TECHNIQUES USED

The columns in the table are **reduced to two dimensions using PCA.** We are going to use an algorithm like **K-Means** to cluster the given data in different clusters based on the average user rating given by the user. Additionally, we are going to predict the average rating a user would give to a new unreleased board game based on its attributes, using **linear regression**. For checking the error between the training and the test sets, we are going to use the **mean squared error** metric.

SOFTWARE USED

Pandas – This library will be used to import the database, given in .csv format, into Pandas DF. Additionally, it is also used for performing other DF operations.

Matplotlib – This library would be mainly used for the plotting of graphs, histograms of the given data.

SKL – SVM is a ML library for the Python programming language. It features various classification, regression and clustering algorithms including SVM, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Anaconda - Anaconda is a freemium open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing. It comes pre-installed with the Pandas, Matplotlib and SVM packages.

REQUIREMENT ANALYSIS WORK

EXPLORING DATASET

The database contains 81312 rows and 20 column attributes. The dataset is from BoardGameGeek, and contains data on 80000 board games. This information was **scraped into csv format** by Sean Beck.

The dataset contains several data points about each board game. Here's a list of the interesting ones:

Name	– name of the board game.
Playing time	– the playing time (given by the manufacturer).
Minplaytime	– the minimum playing time (given by the manufacturer).
Maxplaytime	– the maximum playing time (given by the manufacturer).
minage	– the minimum recommended age to play.
users_rated	– the number of users who rated the game.
average_rating	– the average rating given to the game by users. (0-10)
total_weights	– Number of weights given by users.
average_weight	– the average of all the subjective weights (0-5)

Our data file looks like this (some columns have been removed for readability):-

```
id,type,name,yearpublished,minplayers,maxplayers,playingtime

12333,boardgame,Twilight Struggle,2005,2,2,180

120677,boardgame,Terra Mystica,2012,2,5,150
```

This is in a format called csv, or comma-separated values. Each row of the data is a different board game, and different data points about each board game are **separated by commas within the row**. The first row is the header row, and describes what each data point is. The entire set of one data point, going down, is a column.

We can easily conceptualize a csv file as a matrix:

```
    1       2           3                   4

1   id      type        name                yearpublished

2   12333   boardgame   Twilight Struggle   2005

3   120677  boardgame   Terra Mystica       2012
```

A matrix has some downsides when working with datasets. You can't easily access columns and rows by name, and each column has to have the same datatype. This means that we can't effectively store our board game data in a matrix – the name column contains strings, and the yearpublished column contains integers, which means that we can't store them both in the same matrix.

A DF, on the other hand, can have **different datatypes in each column**. It has a lot of built-in functions for analyzing data as well, such as looking up columns by name.

READING AND ANALYZING THE DATA

Pandas provides data structures and data analysis tools that make manipulating data in Python much quicker and more effective. The most common data structure is called a DF. A DF is an extension of a matrix, as discussed earlier.

We'll now read in our data from a csv file into a Pandas DF, using the read_csv method:-

```
import pandas
games = pandas.read_csv("board_games.csv")
print(games.columns)
```

```
Index(['id', 'type', 'name', 'yearpublished', 'minplayers', 'maxplayers',
       'playingtime', 'minplaytime', 'maxplaytime', 'minage', 'users_rated',
       'average_rating', 'bayes_average_rating', 'total_owners',
       'total_traders', 'total_wanters', 'total_wishers', 'total_comments',
       'total_weights', 'average_weight'], dtype='object')
```

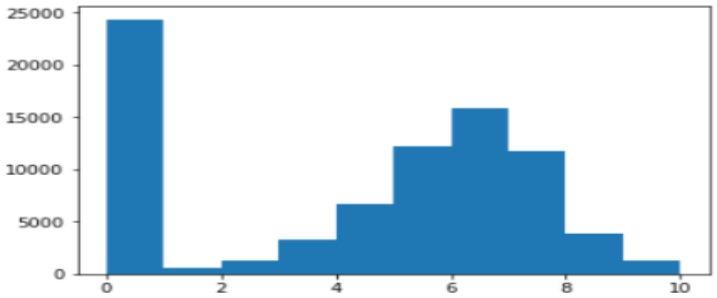
The code above read the data in, and shows us all of the column names. We can also see the shape of the data, which shows that it has 81312 rows, or games, and 20 columns.

```
print(games.shape)
```

```
(81312, 20)
```

If we plot a **histogram** of this column, we can visualize the distribution of ratings. We'll use **Matplotlib** to generate the visualization. Matplotlib is the main plotting infrastructure in Python.

```
import matplotlib.pyplot as plt
plt.hist(games["average_rating"])
plt.show()
```



What we see here is that there are quite a few games with a 0 rating. On closer inspection of the data, we find out that quite a few of the games which have got a 0 rating **don't have any reviews assigned to them**. Hence, we remove all these games so as not to skew our distribution and application. The following command drops all the board games which have no user reviews.

```
games = games[games["users_rated"] > 0]
games = games.dropna(axis=0)
```

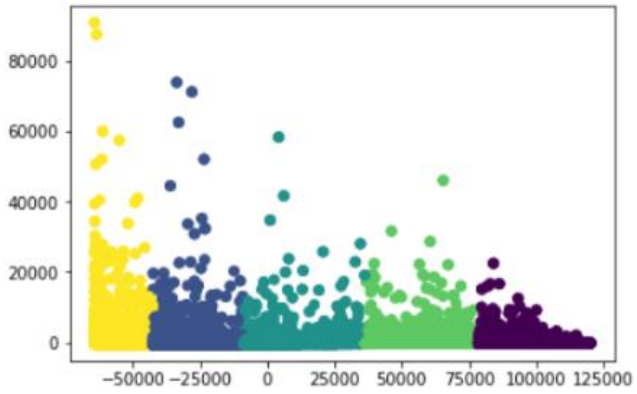
There may be distinct sets of games. One set was the set of games without reviews. Another set could be a set of highly rated games. One way to figure out more about these sets of games is a technique called **clustering**. Clustering enables you to **find patterns within your data easily by grouping similar rows** (in this case, games), together. We'll use a particular type of clustering called **KMC**. SKL has an implementation of KMC that we can use.

```
from sklearn.cluster import KMeans
kmeans_model = KMeans(n_clusters=5, random_state=1)
good_columns = games._get_numeric_data()
kmeans_model.fit(good_columns)
labels = kmeans_model.labels_
```

In order to use the clustering algorithm, we'll first initialize it using two parameters – n_clusters defines how many clusters of games that we want, and random_state is a random seed we set in order to reproduce our results later. We then only get the numeric columns from our DF. Most ML algorithms **can't directly operate on text data**, and can only take numbers as input. Finally, we fit our kmeans model to our data, and get the cluster assignment labels for each row.

We'll have to **reduce the dimensionality of our data**, without losing too much information. One way to do this is a technique called PCA, or PCA. PCA takes multiple columns, and turns them into fewer columns while trying to preserve the unique information in each column. We'll try to turn our board game data into two dimensions, or columns, so we can easily plot it out.

```
from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(good_columns)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=labels)
plt.show()
```



The plot shows us that there are **5 distinct clusters**. We could dive more into which games are in each cluster to learn more about what factors cause games to be clustered.

Since we want to predict average rating, let's see what columns might be interesting for our prediction. One way is to find the **correlation between average_rating and each of the other columns.** We can use the corr method on Pandas DFs to easily find correlations.

games.corr()["average_rating"]

```
Out[15]: id                     0.304201
         yearpublished          0.108461
         minplayers            -0.032701
         maxplayers            -0.008335
         playingtime            0.048994
         minplaytime            0.043985
         maxplaytime            0.048994
         minage                 0.210049
         users_rated            0.112564
         average_rating         1.000000
         bayes_average_rating   0.231563
         total_owners           0.137478
         total_traders          0.119452
         total_wanters          0.196566
         total_wishers          0.171375
         total_comments         0.123714
         total_weights          0.109691
         average_weight         0.351081
         Name: average_rating, dtype: float64
```

We see that the **average_weight and id columns correlate best to rating**. Id's are presumably assigned when the game is added to the database, so this likely indicates that games created later score higher in the ratings. **Average_weight indicates the "depth" or complexity of a game**, so it stands to reason that more complex games are reviewed better.

Before we get started, we should only select the columns that are relevant when training our algorithm. We'll want to **remove certain columns that aren't numeric and columns that can only be computed if you already know the average rating**. Including these columns will destroy the purpose of the classifier, which is to predict the rating without any previous knowledge. Using columns that can only be computed with knowledge of the target can lead to **overfitting**, where your model is good in a training set, but **doesn't generalize well to future data**.

```
columns = games.columns.tolist()
columns = [c for c in columns if c not in ["bayes_average_rating", "average_rating", "type", "name"]]
target = "average_rating"
```

IMPLEMENTATION

TRAINING AND TEST SETS

In order to prevent overfitting, we'll **train our algorithm on a set consisting of 80% of the data, and test it on another set consisting of 20% of the data.** To do this, we first randomly sample 80% of the rows to be in the training set, then put everything else in the testing set.

```
from sklearn.cross_validation import train_test_split
train = games.sample(frac=0.8, random_state=1)
test = games.loc[~games.index.isin(train.index)]
print(train.shape)
print(test.shape)
```

```
(45515, 20)

(11379, 20)
```

FITTING LINEAR REGRESSION

Linear regression only works well when the **predictor variables and the target variable are linearly correlated**. As we saw earlier, a few of the predictors are correlated with the target, so linear regression should work well for us.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(train[columns], train[target])
```

PREDICTING ERROR

Linear regression

After we train the model, we can make predictions on new data with it. This new data has to be in the **exact same format as the training data**, or the model won't make accurate predictions. We select the same subset of columns from the test set, and then make predictions on it.

```
from sklearn.metrics import mean_squared_error
predictions = model.predict(test[columns])
mean_squared_error(predictions, test[target])
```

```
1.8239281903519875
```

Random forest

The random forest algorithm can **find nonlinearities in data that a linear regression wouldn't be able to pick up on**. Predictions made with a random forest usually have less error than predictions made by a linear regression.

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, min_samples_leaf=10, random_state=1)
model.fit(train[columns], train[target])
predictions = model.predict(test[columns])
print(mean_squared_error(predictions, test[target]))
```

```
1.4144905030983794
```

PREDICTIONS

print(predictions)

```
[ 7.87323596  7.7603222  7.87385465  ...,  7.26467783  5.89682253  7.64492849]
```

The predictions are given in a list format. On expanding the list and comparing it with the average rating, one can make out that **the average rating are well within error limits.**

Note that the above predictions is made **on the basis of a linear regression model**. A random forest algorithm will be even more accurate. One can attempt to try various other algorithms to see which one gets the least error. Some other algorithms are SVM, ensembling multiple models etc.

DESIGN

We are using Tkinter in Python to create a GUI for the application. The GUI **accepts the values of different fields from the user, and returns an average rating to the user**.

```
In [ ]: from tkinter import *
        import pandas
        from sklearn.cross_validation import train_test_split
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error

        m=[]
        l=[]
        def printData(name,year,minplayers,maxplayers,playingtime,minplaytime,maxplaytime,minage,users_rated,total_owners,total_trade

            print(name)
            print(year)
            print(minplayers)
            print(maxplayers)
            print(playingtime)
            print(minplaytime)
            print(maxplaytime)
            print(minage)
            print(users_rated)
            print(total_owners)
            print(total_traders)
            print(total_wanters)
            print(total_wishers)
            print(total_comments)
            print(total_weights)
            print(average_weight)
            print(game_id)
            root.destroy()

        def get_input():

            name = entry1.get()
            year = entry2.get()
            minplayers=entry3.get()
            maxplayers=entry4.get()
            playingtime=entry5.get()
            minplaytime=entry6.get()
            maxplaytime=entry7.get()
            minage=entry8.get()
            users_rated=entry9.get()
            total_owners=entry10.get()
            total_traders=entry11.get()
            total_wanters=entry12.get()
            total_wishers=entry13.get()
            total_comments=entry14.get()
            total_weights=entry15.get()
            average_weight=entry16.get()
            game_id=entry17.get()


            m=[year,minplayers,maxplayers,playingtime,minplaytime,maxplaytime,minage,users_rated,total_owners,total_traders,total_wanters
            l.append(m)
            printData(name,year,minplayers,maxplayers,playingtime,minplaytime,maxplaytime,minage,users_rated,total_owners,total_traders,t

        root=Tk()

        label1 = Label(root,text = 'Name of Board Game(default : eclipse)')
        label1.pack()
        label1.config(justify = CENTER)
        entry1=Entry(root,width=30)
        entry1.pack()

        label2 = Label(root,text = 'year published(default : 2011)')
        label2.pack()
        label2.config(justify = CENTER)
        entry2=Entry(root,width=30)
        entry2.pack()

        label3 = Label(root,text = 'minimum players required to play the game(default : 2)')
        label3.pack()
        label3.config(justify = CENTER)
        entry3=Entry(root,width=30)
```

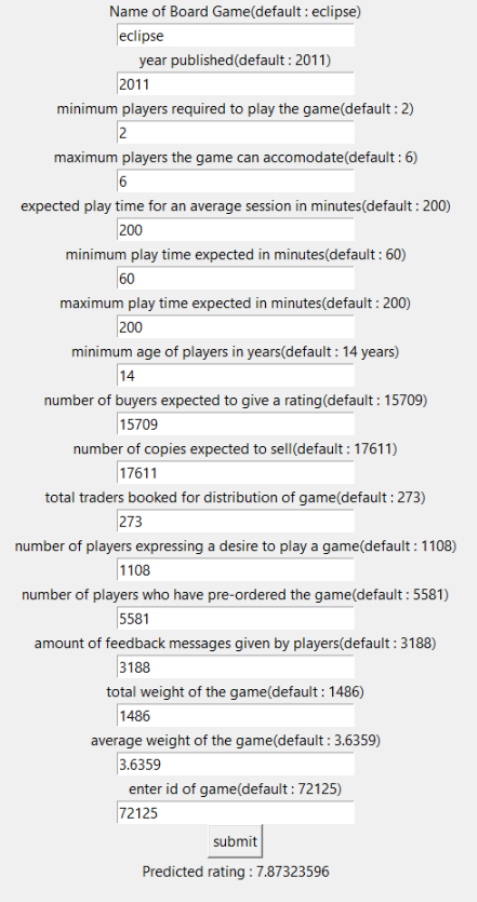
```
label3.config(justify = CENTER)
entry3=Entry(root,width=30)
entry3.pack()

label4 = Label(root,text = 'maximum players the game can accomodate(default : 6)')
label4.pack()
label4.config(justify = CENTER)
entry4=Entry(root,width=30)
entry4.pack()

label5 = Label(root,text = 'expected play time for an average session in minutes(default : 200)')
label5.pack()
label5.config(justify = CENTER)
entry5=Entry(root,width=30)
entry5.pack()

label6 = Label(root,text = 'minimum play time expected in minutes(default : 60)')
label6.pack()
label6.config(justify = CENTER)
entry6=Entry(root,width=30)
entry6.pack()

label7 = Label(root,text = 'maximum play time expected in minutes(default : 200)')
label7.pack()
label7.config(justify = CENTER)
entry7=Entry(root,width=30)
entry7.pack()

label8 = Label(root,text = 'minimum age of players in years(default : 14 years)')
label8.pack()
label8.config(justify = CENTER)
entry8=Entry(root,width=30)
entry8.pack()

label9 = Label(root,text = 'number of buyers expected to give a rating(default : 15709)')
label9.pack()
label9.config(justify = CENTER)
entry9=Entry(root,width=30)
```

```
label9.pack()
label9.config(justify = CENTER)
entry9=Entry(root,width=30)
entry9.pack()

label10 = Label(root,text = 'number of copies expected to sell(default : 17611)')
label10.pack()
label10.config(justify = CENTER)
entry10=Entry(root,width=30)
entry10.pack()

label11 = Label(root,text = 'total traders booked for distribution of game(default : 273)')
label11.pack()
label11.config(justify = CENTER)
entry11=Entry(root,width=30)
entry11.pack()

label12 = Label(root,text = 'number of players expressing a desire to play a game(default : 1108)')
label12.pack()
label12.config(justify = CENTER)
entry12=Entry(root,width=30)
entry12.pack()

label13 = Label(root,text = 'number of players who have pre-ordered the game(default : 5581)')
label13.pack()
label13.config(justify = CENTER)
entry13=Entry(root,width=30)
entry13.pack()

label14 = Label(root,text = 'amount of feedback messages given by players(default : 3188)')
label14.pack()
label14.config(justify = CENTER)
entry14=Entry(root,width=30)
entry14.pack()

label15 = Label(root,text = 'total weight of the game(default : 1486)')
label15.pack()
label15.config(justify = CENTER)
entry15=Entry(root,width=30)
```

```
label14 = Label(root,text = 'amount of feedback messages given by players(default : 3188)')
label14.pack()
label14.config(justify = CENTER)
entry14=Entry(root,width=30)
entry14.pack()

label15 = Label(root,text = 'total weight of the game(default : 1486)')
label15.pack()
label15.config(justify = CENTER)
entry15=Entry(root,width=30)
entry15.pack()

label16= Label(root,text = 'average weight of the game(default : 3.6359)')
label16.pack()
label16.config(justify = CENTER)
entry16=Entry(root,width=30)
entry16.pack()

label17= Label(root,text = 'enter id of game(default : 72125)')
label17.pack()
label17.config(justify = CENTER)
entry17=Entry(root,width=30)
entry17.pack()

button1=Button(root,text='submit')
button1.pack()
button1.config(command=get_input)


test_example=pandas.DataFrame(l)
predictions = model.predict(test_example)
label18= Label(root,text = 'Predicted rating :',predictions)
label18.pack()
label18.config(justify = CENTER)

root.mainloop()
```

TESTING

For the purposes of this test, we have used a board game from the test set of the database. **The actual average rating for the board game was 8.07933**, well within the mean squared error limits of linear regression. This error can be brought down further by changing the classifier to random forest.



CONCLUSION

We have successfully predicted the average rating of the board game database, using linear regression, and compared it to the random forest algorithm.

We managed to implement basic ML techniques like **KMC, MSE and PCA dimensionality reduction** to analyze the data.

We noticed that errors and predictions could be made more accurate by **choosing datasets with more important attributes and less redundant ones** and **implementing other algorithms which may fit the dataset better.**

As one can see, this project can be extrapolated to other media, which work on a user rating basis. It can be used to focus test various products and services, before the company makes a mistake and spends a significant amount of money on a failed product.

In the future, it's possible to expand the functionality of this project as follows:-

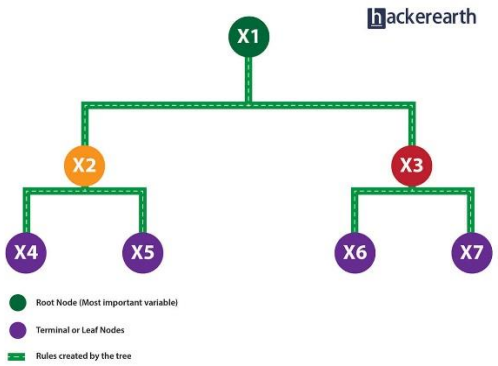
- Implementing SVM algorithm
- Trying to ensemble multiple models to create better predictions.
- Trying to predict a different column, such as average_weight.
- Generate features from the text, such as length of the name of the game, number of words, etc.

APPENDICES

APPENDIX I

Random Forests

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.



APPENDIX II

PCA

PCA is a method of extracting important variables (in form of components) from a large set of variables available in a data set. It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible. With fewer variables, visualization also becomes much more meaningful. PCA is more useful when dealing with 3 or higher dimensional data. It is always performed on a symmetric correlation or covariance matrix. This means the matrix should be numeric and have standardized data.

REFERENCES

1. Delvin, Josh., Rice, Charles. & Sundby, Eric, 2017. Data Science Blog: Dataquest. < https://www.dataquest.io/blog/>

2. Beck, Sean, 2017. <https://github.com/ThaWeatherman/scrapers>. Scrapers repository.

3. Numpy, accessed Oct 1st 2017. <http://www.numpy.org/>

4. Pandas, accessed Oct 1st 2017. < http://pandas.pydata.org/pandas-docs/stable/indexing.html>

5. Hall, M.A. (2000). Correlation-based feature selection of discrete and numeric class ML. (Working paper 00/08). Hamilton, New Zealand: University of Waikato, Department of Computer Science.