FOCUS TESTING

##### 15IT322E- PYTHON PROGRAMMING PROJECT REPORT

###### ***Submitted by***

##### S.MAHADEVAN (RA1511003010422)

*For the assessment of Semester V Minor Project*

# srm newlogo

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**

**CHAPTER NO. TITLE PAGE NO.**

ABSTRACT 1

LIST OF TABLE 3

LIST OF FIGURES 4-5

LIST OF SYMBOLS 6

**1 INTRODUCTION 7-8**

1.1 GENERAL 7

1.2 TECHNIQUES USED 8

1.3 SOFTWARE USED 8

**2 REQUIREMENT ANALYSIS 9-15**

* 1. EXPLORING DATASET 9-10
  2. READING & ANALYZING 11-15

DATA

**3 IMPLEMENTATION 16-18**

* 1. TRAINING AND TEST SETS 16
  2. FITTING LINEAR REGRESSION 16
  3. PREDICTING ERROR 17

3.3.1 LINEAR REGRESSION 17

3.3.2 RANDOM FOREST 17

3.4 PREDICTIONS 18

**4 DESIGN 19-21**

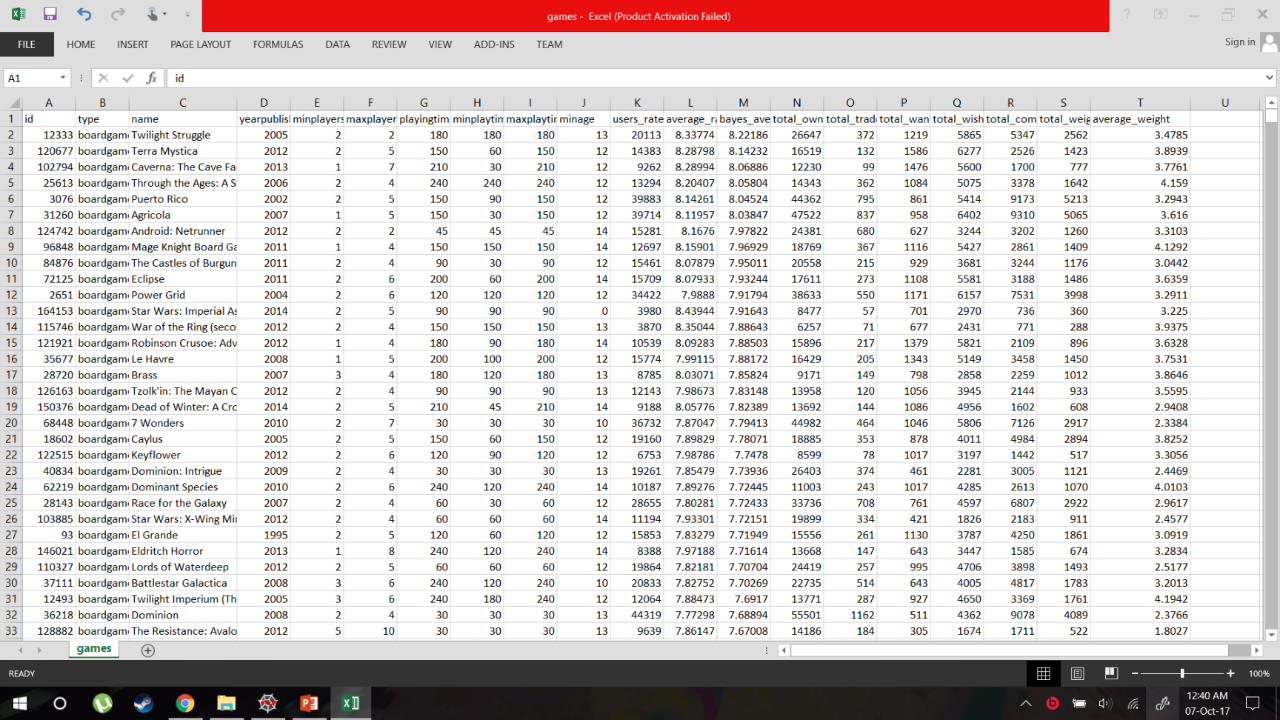
**5 TESTING 22**

**6 CONCLUSION 23**

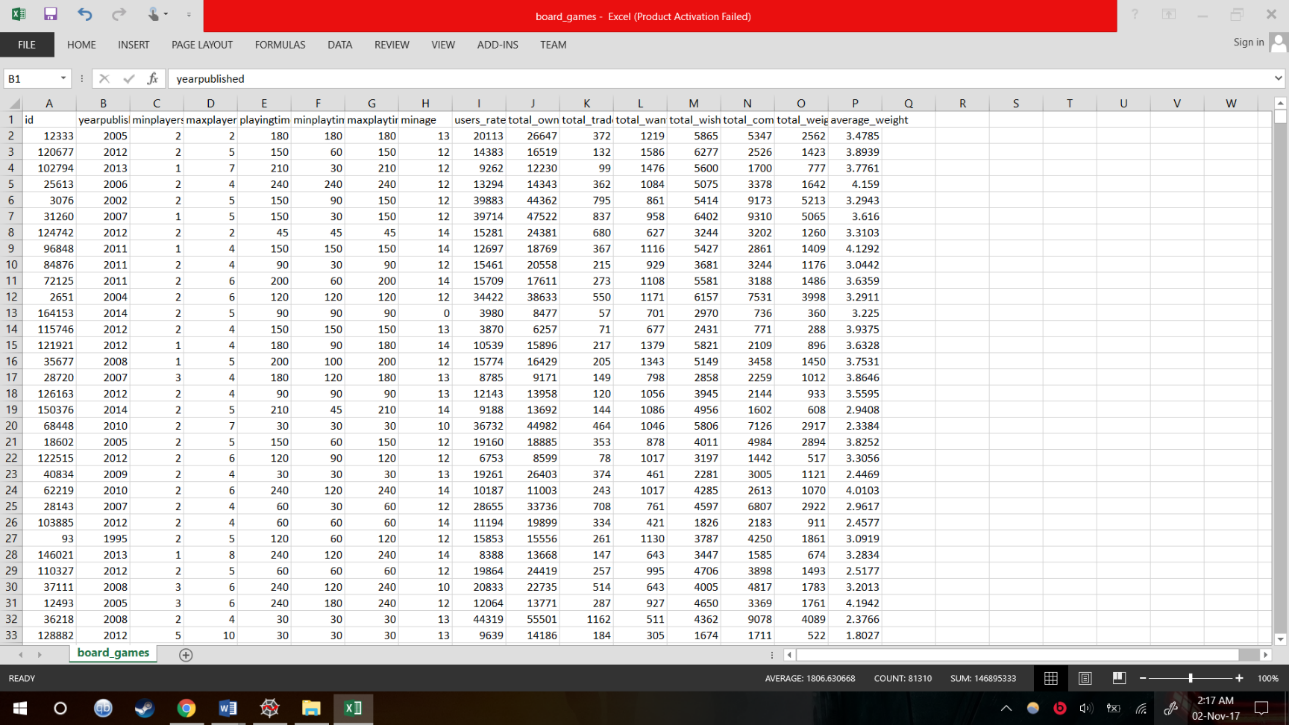
**7 APPENDICES 24**

**8 REFERENCES 25**

**LIST OF TABLES**

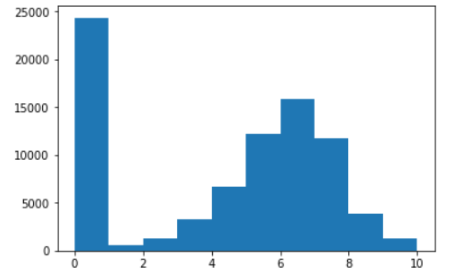


A snapshot of the board game database to be used for the project

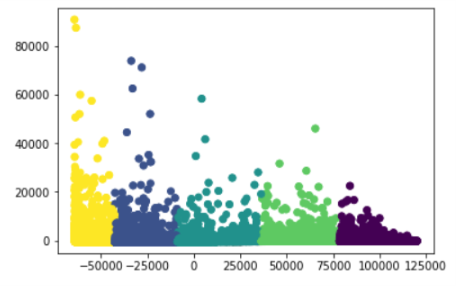


The final table to be used for testing, with all redundant columns removed

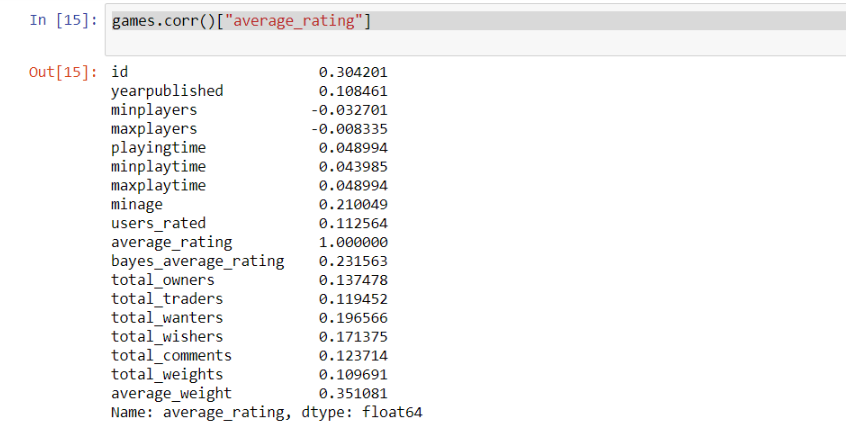
**LIST OF FIGURES**



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



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)*

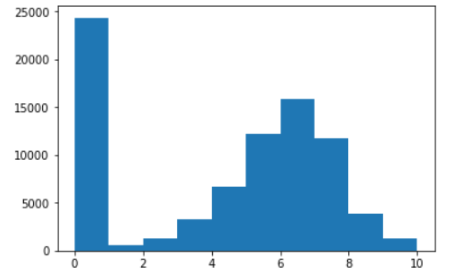
(81312, 20)

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.

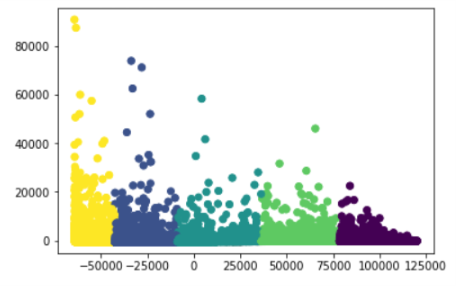
*from sklearn.decomposition import PCA*

*pca\_2 = PCA(2)*

*plot\_columns = pca\_2.fit\_transform(good\_columns)*

*plt.scatter(x=plot\_columns[:,0], y=plot\_columns[:,1], c=labels)*

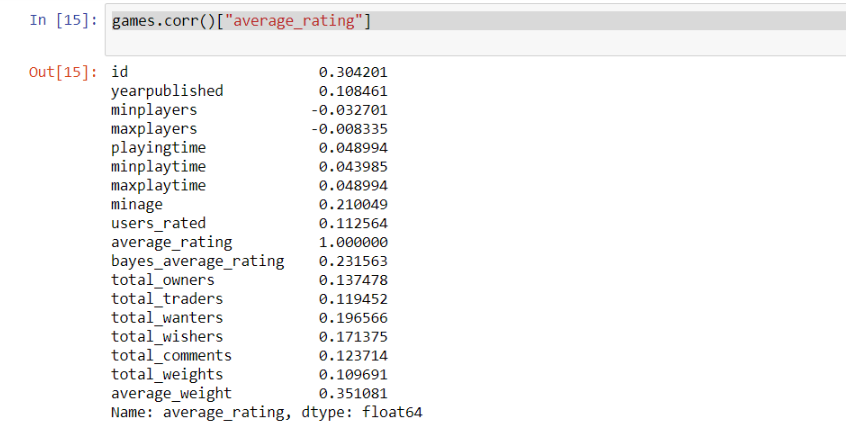
*plt.show()*



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"]*



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)*

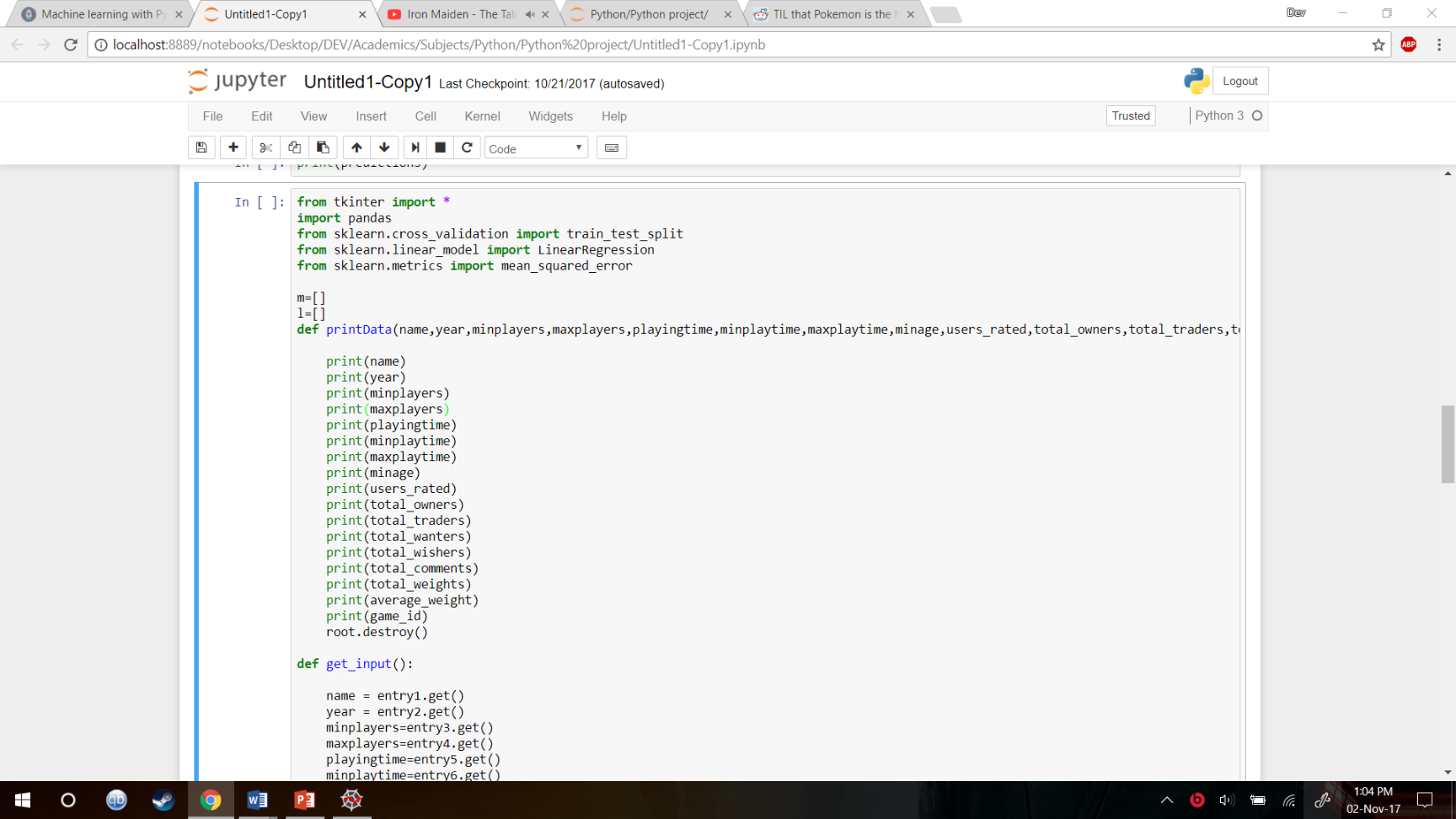
[ 7.87323596 7.7603222 7.87385465 ..., 7.26467783 5.89682253 7.64492849]

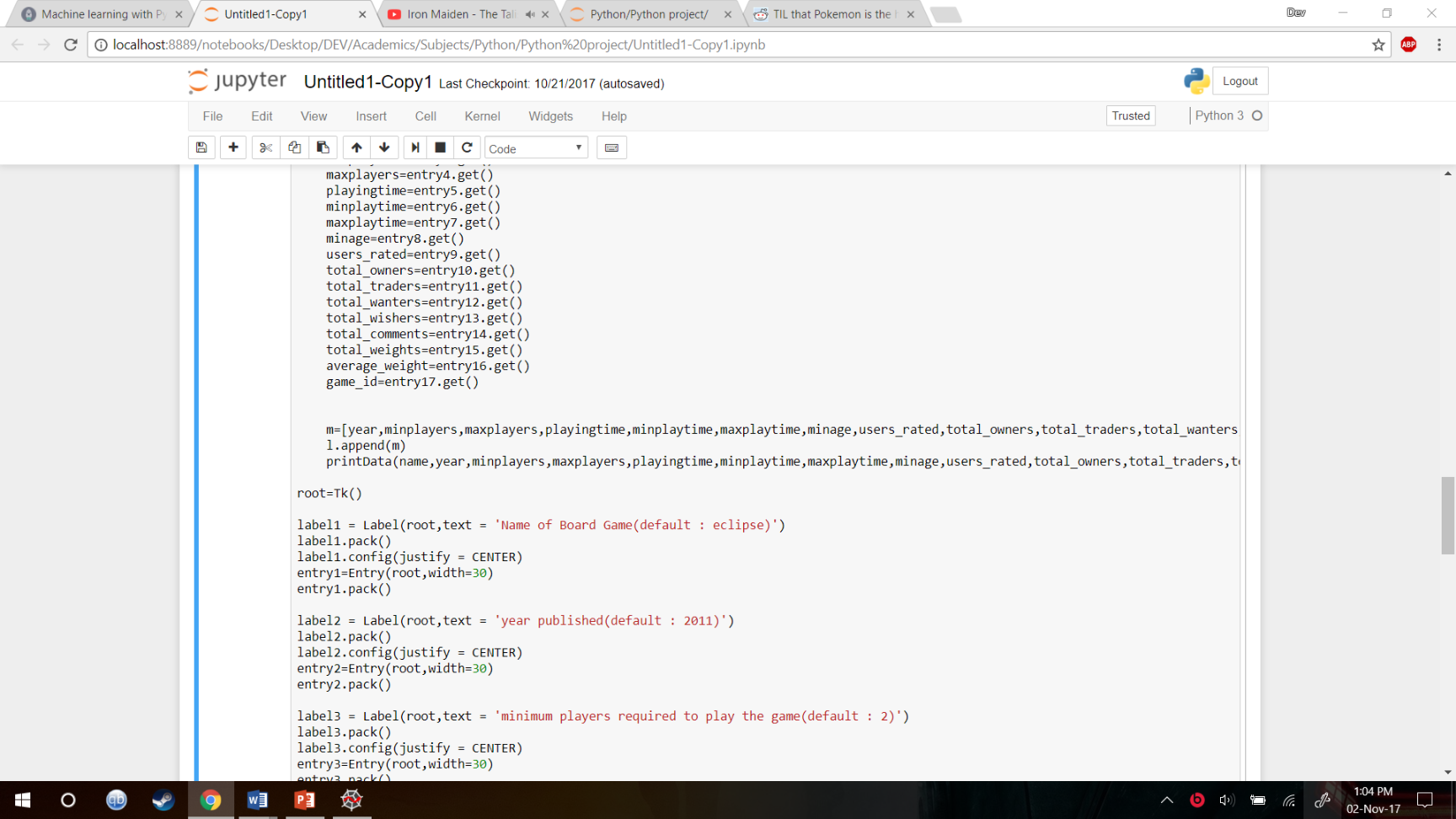
The predictions are given in a list format. On expanding the list and comparing it with the average rating, one can make out that **the average rating are well within error limits.**

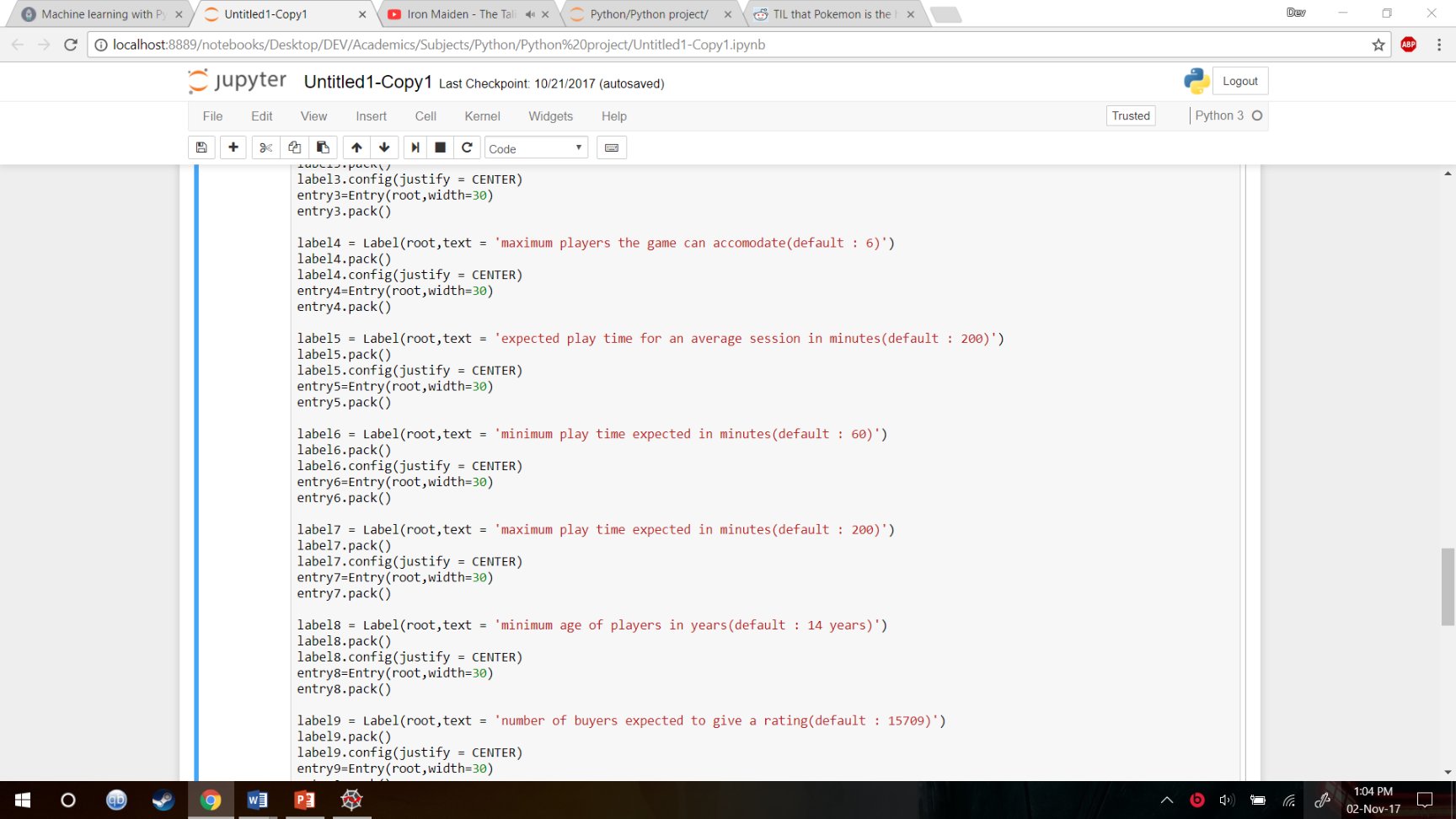
Note that the above predictions is made **on the basis of a linear regression model**. A random forest algorithm will be even more accurate. One can attempt to try various other algorithms to see which one gets the least error. Some other algorithms are SVM, ensembling multiple models etc.

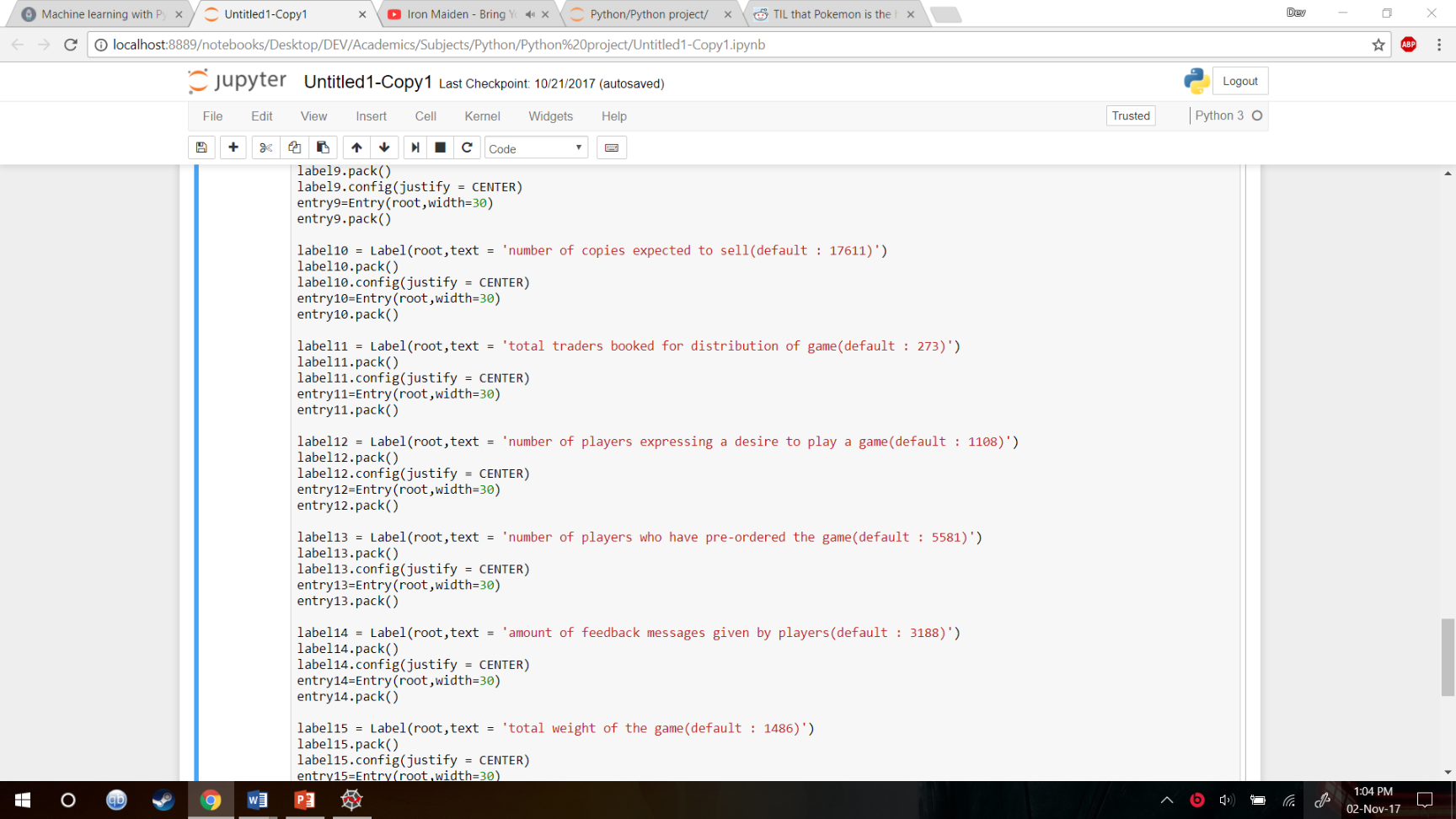
**DESIGN**

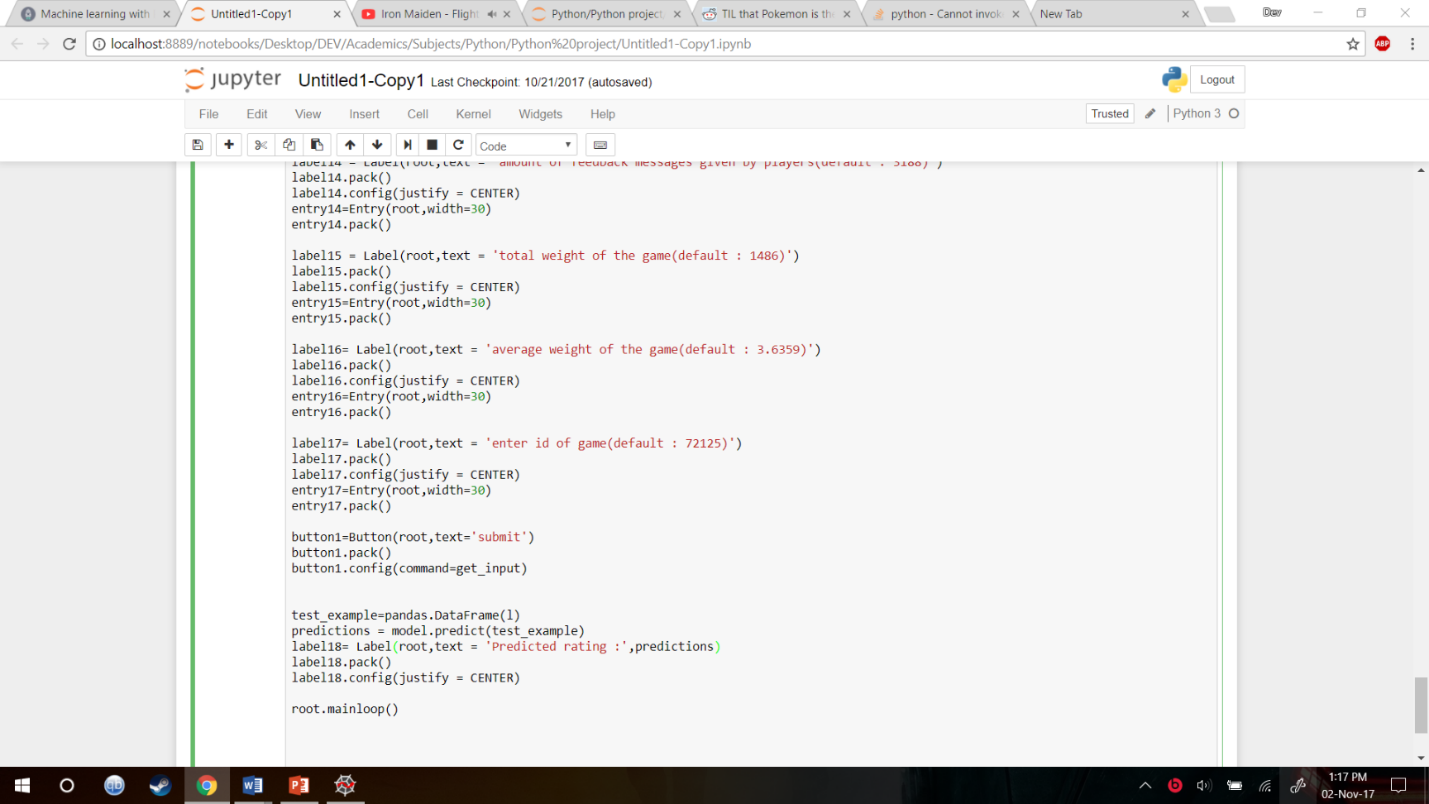
We are using Tkinter in Python to create a GUI for the application. The GUI **accepts the values of different fields from the user, and returns an average rating to the user**.





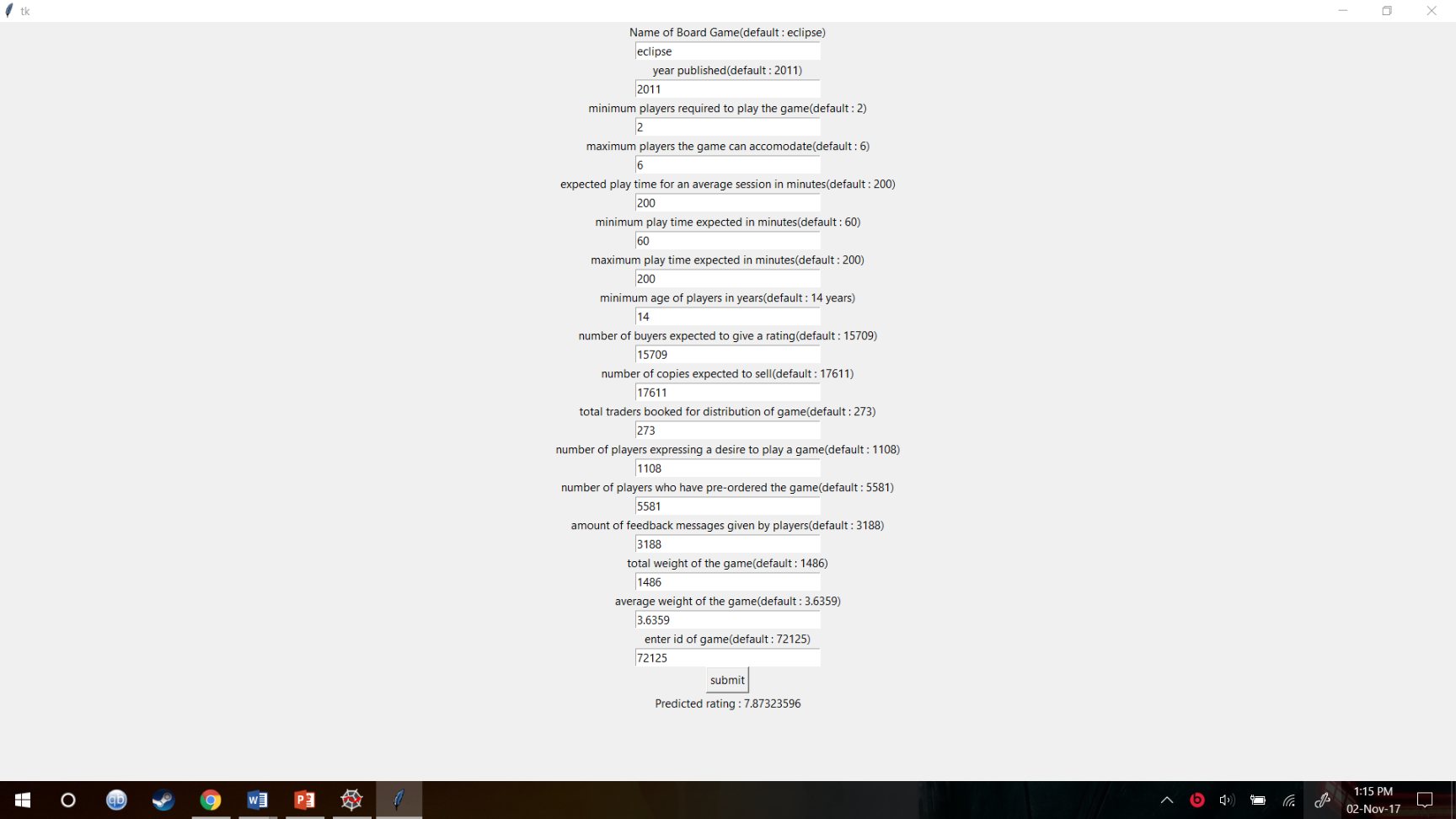






**TESTING**

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:-

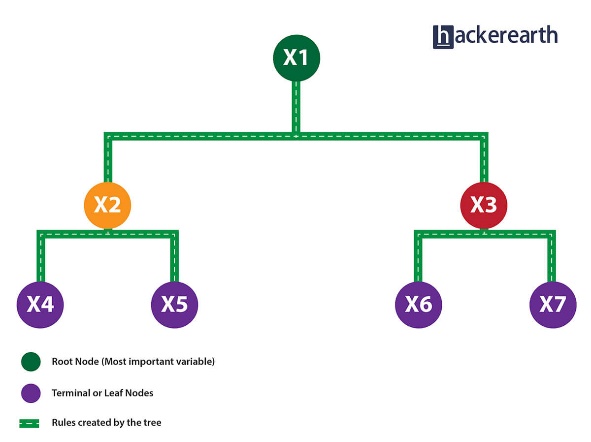
* Implementing SVM algorithm
* Trying to ensemble multiple models to create better predictions.
* Trying to predict a different column, such as average\_weight.
* Generate features from the text, such as length of the name of the game, number of words, etc.

**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.